A Goal-Directed Implementation of 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

submitted 1 June 2010; revised 18 October 2010; accepted 25 July 2012

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 Type-1 ($\mathcal{ALCQ}$) theory.

KEYWORDS: Knowledge representation, well-founded semantics, description logics, implementation.

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.

* This work was partially supported by project ERRO (PTDC/EIA-CCO/121823/2010). The first author was supported by the grant SFRH/BD/64038/2009.
Example 1
Consider a scenario application for a Customs Agency where an ontology is used to assess and classify aspects of imports and exports. An ontology with such characteristics would embrace several thousand axioms. In these axioms, as an example, one can define several axioms about countries (e.g. Scandinavian countries are European countries; Norway is a Scandinavian country; etc). Moreover, one can also model in the ontology knowledge that is certain, in the sense that it does not allow for exception. With such certain knowledge, first-order logics deduction is what is desired. For example, in this scenario, one could state that Scandinavian countries are always considered safe countries. All of this can be easily represented by an ontology defined in a Description Logic (DL) language (Baader et al. 2007)

\begin{align*}
\text{ScandinavianCountry} & \sqsubseteq \text{EuropeanCountry} \\
\text{Norway} : \text{ScandinavianCountry} \\
\text{ScandinavianCountry} & \sqsubseteq \text{SafeCountry}
\end{align*}

(1)

Besides these axioms about countries, one may want to specify some additional knowledge, e.g. for defining conditions about whether or not to inspect entering shipments based on their country of origin. For example, we may want to state that one should inspect any vessel containing a shipment coming from a country that is not guaranteed to be safe. Note here that the closed world assumption is needed to define such a statement. In fact, intuitively one wants to assume that, if one is unsure whether given country is safe, then (at least for the sake of this statement) the country should be considered unsafe and an inspection performed. When using description logic with classical negation this behavior would not be obtained, as a condition \( \neg \text{SafeCountry}(\text{country}) \) would only be true in case one knows for sure that the country is not safe. However, this statement can be easily expressed by using a (non-monotonic) rule with default negation as in (2).

\begin{align*}
\text{inspect}(X) \leftarrow \text{hasShipment}(X, \text{Country}), \quad \text{not safeCountry(Country)}. 
\end{align*}

(2)

Such rules are non-monotonic in the sense that further knowledge – in this case about safety of countries – can invalidate previous conclusions about inspection. This kind of non-monotonic rule is quite useful to specify default knowledge that may be subject to exceptions. In this sense rule (2) can be seen as stating that by default shipments should be inspected, but an exception to this default rule are shipments coming from safe countries.

Note that defining some statements about a predicate using default rules does not necessarily mean that all statements defining that predicate should be default rules. For example, we may want to state that diplomatic shipments are not subject to inspection, regardless their country of origin. Here no default reasoning is involved.

---

1 We write DL formulas with the usual notation and the usual DL operators (see Baader et al. 2007): so that the argument variable of a unary predicate is not displayed and the first letter of the predicate’s name is capitalized.

2 We write rules in the usual notation for logic programming, where predicates names are non-capitalized (even if common to the ontology, where they are capitalized), and variables are capitalized and implicitly universally quantified in the rule head.
Moreover, negation here is not the default negation of logic programming, but rather
classical negation. Such a statement could be modeled using normal logic programs
extended with explicit negation (Gelfond and Lifschitz 1990) with the rule:
\[
\neg \text{inspect}(X) \leftarrow \text{diplomaticShipment}(X)
\] (3)

or, having the possibility to use an ontology formalized as a fragment of first-order
logic, with a statement:
\[
\text{DiplomaticShipment} \sqsubseteq \neg \text{Inspect}
\] (4)

However, these two representations of the statement above are not equivalent. In
particular, with axiom 4 one can conclude that a given shipment is not diplomatic
whenever one knows, by some other rule, that it should be inspected; this is not
the case with rule 3. The behavior of rules with explicit negation can be easily
modeled by using ontologies; for that it suffices to create a new concept standing
for the explicit negation, say NonInspect, replacing the explicitly negated literal in
the rule by this new concept, and adding the axiom NonInspect \(\equiv \neg \text{Inspect}\). On
the contrary, with programs extended with explicit negation one cannot obtain the
full expressivity that is obtained with some of the DL-based languages for ontolo-
gies, simply because some of these languages belong to a higher complexity class.
Another important feature not supported by explicit negation but typically sup-
ported by DL-based ontologies, is the possibility of using existential quantification
as illustrated in Example 2 below.

The foregoing example briefly illustrates some advantages of combining features
of ontologies with features of logic programming-like rules, and of doing so in a way
that allows knowledge about instances to be fully inter-definable between rules and
an ontology (as happens with the predicate inspect/1 above, which is both defined
in the rules and the ontology).

In fact, this combination of ontology and rule languages has gained particular
importance in the context of the Semantic Web (Horrocks et al. 2006). In this
context, a family of languages for representing background knowledge in the Web,
OWL 2 (Hitzler et al. 2009), has been recommended by the World Wide Web
Consortium (W3C). OWL 2 languages are based on Description Logics (Baader
et al. 2007), which, as in the example above, employ open-world assumption. In
addition, the rule interchange language RIF (Morgenstern et al. 2010) has recently
been formally recommended to the W3C. RIF has many similarities with the logic
programming language used in the example above, and adopts the closed-world
assumption.

The existence of both rules and ontology languages should make it possible to
combine open and closed world reasoning, and this combination is indeed important
in several domains related to the Semantic Web. As a further example where this
combination is desired, consider the large case study described in (Patel et al. 2007),
containing millions of assertions concerning matching patient records to clinical
trials criteria. In this case study, open world reasoning is needed in deductions
about domains such as radiology and laboratory data: unless a lab or radiology test
asserts a negative finding no arbitrary assumptions about the results of the test can
be made (e.g. we can only be certain that some patient does not have a specific kind
of cancer if the corresponding test is known to have negative result). However, as
observed in (Patel et al. 2007), closed world reasoning can and should be used with
pharmacy data to infer that a patient is not on a medication if this is not asserted.
The work of (Patel et al. 2007) applies only open world reasoning but claims that
the usage of closed world reasoning in pharmacy data would be highly desirable.
Similar situations occur e.g. in matchmaking using Semantic Web Services (cf.
(Grimm and Hitzler 08)), where again a combination of ontology languages relying
on open-world reasoning, with rule languages relying on closed-world reasoning is
considered highly desirable.

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 (Gelfond and Lifschitz 1990) or the well-founded semantics (WFS) (van
Gelder et al. 1991), 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 (Baral 2002). Eiter et al. 2004a; Swift 2004; Motik 2006). On
the other hand, WFS is weaker than the answer-set semantics (in the sense that
it is more skeptical), but has the advantages of lower complexity and its ability
to be evaluated in a query-oriented fashion, which have led to its integration into
Prolog systems. Another possibility of maintaining the complexity under reasonable
bounds for Semantic Web applications, is to limit the expressivity of both the
ontological part of the knowledge base, and of the rules. For instance, (Calì et al.
2012) considers variants of Datalog that allow for existential quantification in rule
heads along with other features to support a restricted form of ontological reasoning,
yet restrict the rule syntax to obtain tractability.

Keeping to the general form of logic programming rules, but maintaining the
complexity of reasoning under reasonable bounds, several formalisms have con-
cerned themselves with combining ontologies with general WFS rules (Drabent and
Maluszynski 2007; Eiter et al. 2004b; Knorr et al. 2008). Among these, the Well-
Founded Semantics for Hybrid MKNF knowledge bases (MKNF_{WFS}), introduced
in (Knorr et al. 2008) and overviewed in Section 2 below, is the only one that 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 (Motik and Rosati 2007) and, as in (Motik and Rosati 2007), allows the knowl-
edge base to have both closed- and open-world (classical) negation. For comparisons
of MKNF_{WFS} with the approaches for combining rules with ontologies mentioned
above, see (Knorr et al. 2011).

Example 2
The following fragment, adapted from an example in (Motik and Rosati 2007),
concerning car insurance premiums illustrates several properties of MKNF_{WFS}.
The ontology consists of the following axioms, which state that Married and
NonMarried are complementary concepts, that anyone who is not married is high risk, and that anyone with a spouse is married:

\[
\text{NonMarried} \equiv \neg \text{Married} \\
\neg \text{Married} \sqsubseteq \text{HighRisk} \\
\exists \text{Spouse}. \top \sqsubseteq \text{Married}.
\]

The rule base consists of the following rules, which state that anyone who is not known to be married is to be assumed to be non-married, that those who are known to be high-risk should have a surcharge, and that those that have a known spouse should have a discount:

\[
\text{K nonMarried}(X) \leftarrow \text{K person}(X), \text{not married}(X).
\]

\[
\text{K surcharge}(X) \leftarrow \text{K highRisk}(X), \text{K person}(X).
\]

\[
\text{K discount}(X) \leftarrow \text{K spouse}(X,Y), \text{K person}(X), \text{K person}(Y).
\]

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. 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)). Intuitively, Kϕ stands for “ϕ is known to be true”, whereas not ϕ stands for “ϕ is not known to be true”.

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, ¬married(john) would hold, and by the second axiom highRisk(john) would also hold. By the last rule, surcharge(john) would hold as well. 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, not related to john either through rules nor axioms in the ontology, do not need to be considered.

Now suppose that one learns that a person named Bill has a spouse, and that Bob is the spouse of Ann. This can be formalized by adding the corresponding facts for the predicate person, the fact spouse(bob, ann) and the DL assertion bill : \( \exists \text{Spouse}. \top \). In this case one would expect that neither Bill nor Bob are considered high risk, and so should not have a surcharge; and that since Bob (contrary to Bill) has a known spouse, he should have a discount. This is in fact the result of MKNF_{WFS}. Note that, it is only possible to represent the difference between the situations of Bill and Bob by using the existential quantification of DL, something that is not possible in logic programming alone.

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 (\( ? \)). SLG(\( \mathcal{O} \)) is sound and complete, as well as terminating for various classes of programs (e.g. datalog).
In addition SLG(\(O\)) is relevant in the sense of Example 2, i.e. in general one does not need to compute the whole model (for every object in the knowledge base) to answer a specific query. This relevancy is a critical requirement for scalability in numerous practical applications (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 to \(I\) through rules or axioms. This is also clear e.g. in the above mentioned case study about matching patient records for clinical trials criteria (Patel et al. 2007). In this study one is interested in finding out whether a given patient matches the criteria for a given trial or, at most, which patients match a given criteria; for scalability, it is crucial that such a query does not need to go over all patients and all criteria. In a similar manner, when assessing a shipment in Example 1 it is infeasible to have to consider all shipments into a country: the query must be relevant to be useful.

SLG(\(O\)) 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 prototype query evaluator\(^3\) for MKNF\(_{WFS}\), called CDF-Rules, which fixes the ontology part to ALCQ theories, and makes use of the prover from XSB’s ontology management library, the Coherent Description Framework (CDF) (Swift and Warren 2003) (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 (Motik and Rosati 2007), are essentially formulas in the logics of minimal knowledge and negation as failure (MKNF) (Lifschitz 1991), i.e. first-order logics with equality represented as the congruence relation \(\approx\), and two modal operators \(K\) and \(\text{not}\), which allow inspection of the knowledge base. Intuitively, given a first-order formula \(\varphi\), \(K\varphi\) asks whether \(\varphi\) is known, while \(\text{not}\varphi\) checks whether \(\varphi\) 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 \(O\) 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 called concepts) \(N_C\). An atom \(P(t_1,\ldots,t_n)\), where \(P \in N_C\)

\(^3\) The implementation is available from the XSB CVS repository as part of the CDF package in the subdirectory packages/altCDF/mknf.
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 \( r \) is of the form

\[
\text{K} \quad H_i \leftarrow \text{K} \quad A_1, \ldots, \text{K} \quad A_n, \text{not} \quad B_1, \ldots, \text{not} \quad B_m
\]

where \( H_i, A_i, \) and \( B_i \) are atoms. \( H \) is called the (rule) head and the sets \( \{ \text{K} \quad A_i \} \), and \( \{ \text{not} \quad B_j \} \) form the (rule) body. Atoms of the form \( \text{K} \quad A \) are also called positive literals or modal \( \text{K} \)-atoms while atoms of the form \( \text{not} \quad A \) are called negative literals or modal 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 \( (\mathcal{O}, \mathcal{P}) \).

In Definition 1, the modal operators of MKNF logics are only applied to atoms in the rules, which might indicate that it would suffice to use a simpler logic that does not deal with the application of the modal operators to complex formulae. However, for defining the meaning of these hybrid knowledge bases, modal operators do need to be applied to complex formulas, namely to the whole ontology \( \mathcal{O} \) (cf. Definition 2). Intuitively, a hybrid knowledge base specifies that: for each rule of the form (5) in \( \mathcal{P} \), the atom \( H \) is known if all the \( A_i \) are known and none of the \( B_j \) is known; and that the whole ontology \( \mathcal{O} \) is known (i.e. \( \text{K} \quad \mathcal{O} \)). As such, using the more general MKNF logic eases the definition of the semantics of hybrid knowledge bases, so similarly to (Motik and Rosati 2007) and (Knorr et al. 2008), we resort to the full MKNF logic (Lifschitz 1991).

For decidability DL-safety is assumed, 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 \( \text{K} \quad B \) 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 (Motik and Rosati 2007)).

The well-founded MKNF semantics, \( \text{MKNF}_{\text{WFS}} \), presented in (Knorr et al. 2008, and further developed in (Knorr et al. 2011), 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 (Knorr et al. 2008): basically the atoms that are true or false. For that reason, and in correspondence to logic programming, we name this partition the well-founded model. First, we recall some notions from (Knorr et al. 2008) which will be useful in the definition of the operators for obtaining that well-founded model.

\[\text{Definition 2}\]

Consider a hybrid MKNF knowledge base \( \mathcal{K} = (\mathcal{O}, \mathcal{P}) \). The set of \( \text{K} \)-atoms of \( \mathcal{K} \), written \( \text{KA}(\mathcal{K}) \), is the smallest set that contains (i) all modal \( \text{K} \)-atoms occurring in \( \mathcal{P} \), and (ii) a modal atom \( \text{K} \quad \xi \) for each modal atom \( \text{not} \quad \xi \) occurring in \( \mathcal{K} \).
Furthermore, for a set of modal atoms $S$, $S_{DL}$ is the subset of DL-atoms of $S$ (Definition 1), and $\hat{S} = \{\xi \mid \text{K} \xi \in S\}$.

Basically $KA(K)$ collects all modal atoms of predicates appearing in the rules, and $\hat{S}$ just removes K operators from the argument set $S$.

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 that represent first-order false DL atoms, along with a program transformation making these new modal atoms available for reasoning with the respective rules.

**Definition 3**

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

In $K^+$, $NP$ represents $\neg P$ (with its corresponding arguments) while $K^*$ 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. For example, to guarantee in Example 1 that proving falsity of $\text{Inspect}$ for some shipment (via the ontology) enforces default negation of $\text{Inspect}$ for that shipment, one would build $K^+$ by adding the axiom $\neg \text{Inspect} \sqsubseteq N\text{Inspect}$ (where $N\text{Inspect}$ is a new symbol not appearing elsewhere), and in $K^*$ all original rules with $\text{inspect}(X)$ in the head would be transformed by adding $\text{not } n_{\text{inspect}}(X)$ to the body. In this case, the rule in (2) would be transformed to:

$$K \text{ inspect}(X) \leftarrow K \text{ hasShipment}(X, \text{Country}), \text{not safeCountry}(\text{Country}), \text{not } n_{\text{inspect}}(X).$$

We continue by recalling the definition in (Knorr et al. 2008) of an operator $T_K$ to allow conclusions to be drawn from positive hybrid MKNF knowledge bases (i.e. knowledge bases where rules have no default negation).

**Definition 4**

For $K$ a positive DL-safe hybrid MKNF knowledge base, $R_K$, $D_K$, and $T_K$ are defined on subsets of $KA(K^*)$ as follows:

$$R_K(S) = S \cup \{\text{K } A_i \in S \mid \text{K contains a rule of the form (1) such that } \text{K } A_i \in S \text{ for each } 1 \leq i \leq n\}$$

$$D_K(S) = \{\text{K } \xi \mid \text{K } \xi \in KA(K^*) \text{ and } O \cup \hat{S}_{DL} \models \xi \} \cup \{\text{K } Q(b_1, \ldots, b_n) \mid \text{K } Q(a_1, \ldots, a_n) \in S \setminus S_{DL}, \text{K } Q(b_1, \ldots, b_n) \in KA(K^*), \text{ and } O \cup \hat{S}_{DL} \models a_i \approx b_i \text{ for } 1 \leq i \leq n\}$$

$$T_K(S) = R_K(S) \cup D_K(S)$$

Note that $K^+$ and $K^*$ are still hybrid MKNF knowledge bases, so we only refer to $K^+$ and $K^*$ explicitly when it is necessary.
$R_K$ derives consequences from the rules in a way similar to the classical $T_P$ operator of definite logic programs, while $D_K$ obtains knowledge from the ontology $O$, both from non-DL-atoms and the equalities occurring in $O$, where the $\approx$ operator defines a congruence relation between individuals.

The operator $T_K$ is shown to be monotonic in (Knorr et al. 2008) so, by the Knaster-Tarski theorem, it has a unique least fixed point, denoted $\text{lfp}(T_K)$, which is reached after a finite number of iteration steps.

The computation of the well-founded models follows the alternating fixed point construction (van Gelder 1989) of the well-founded semantics for logic programs. This approach requires turning a hybrid MKNF knowledge base into a positive one to make $T_K$ applicable.

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

The above transformation resembles that used for answer-sets (Gelfond and Lifschitz 1990) of logic programs and the following two operators are defined.

Definition 6
Let $K = (O, P)$ be a DL-safe hybrid MKNF knowledge base and $S \subseteq KA(K^*)$. We define:

$$\Gamma_K(S) = \text{lfp}(T_K^*/S), \quad \text{and} \quad \Gamma_K'(S) = \text{lfp}(T_K^*/S)$$

Both operators are shown to be antitonic (Knorr et al. 2008), hence their composition is monotonic and form the basis for defining the well-founded MKNF model. Here we present its alternating computation.

$$T_0 = \emptyset \quad \text{and} \quad TU_0 = KA(K^*)$$

$$T_{n+1} = \Gamma_K(TU_n) \quad \text{and} \quad TU_{n+1} = \Gamma_K'(T_n)$$

$$T_{\omega} = \bigcup T_n \quad \text{and} \quad TU_{\omega} = \bigcap TU_n$$

Note that by finiteness of the ground knowledge base (Definition 1) the iteration stops before reaching $\omega$. It was shown in (Knorr et al. 2008) that the sequences are monotonically increasing and decreasing respectively, and that $T_\omega$ and $TU_\omega$ form the well-founded model in the following sense:

Definition 7
Let $K = (O, P)$ be a DL-safe hybrid MKNF knowledge base and let $T_K, TU_K \subseteq KA(K)$, with $T_K$ being $T_\omega$ and $TU_K$ being $TU_\omega$ both restricted to the modal atoms occurring in $KA(K)$. Then

$$M_{WF} = \{K A \mid A \in T_K\} \cup \{K \pi(O)\} \cup \\{\text{not} \ A \mid A \in KA(K) \setminus TU_K\}$$

is the well-founded MKNF model of $K$, where $\pi(O)$ denotes the first-order logic formula equivalent to the ontology $O$ (for details on the translation of $O$ into first-order logic see (Motik and Rosati 2007)).
All modal $K$-atoms in $M_{WF}$ are true, all modal not-atoms are false and all other modal atoms from $KA(K)$ are undefined.

As shown in (Knorr et al. 2008), the well-founded model is sound with respect to the original semantics of (Motik and Rosati 2007), i.e. all atoms true (resp. false) in the well-founded model are also true (resp. false) in the model of (Motik and Rosati 2007). In fact, the relation between the semantics of (Knorr et al. 2008) and (Motik and Rosati 2007), is tantamount to that of the well-founded semantics and the answer-sets semantics of logic programs. Moreover, this definition is in fact a generalization of the original definition of the well-founded semantics of normal logic programs, in the sense that if the ontology is empty then this definition exactly yields the well-founded models according to (van Gelder et al. 1991). For more properties, as well as motivation and intuitions on $MKNF_{WF}$, the reader is referred to (Knorr et al. 2008).

3 XSB Prolog and the Coherent Description Framework

Our implementation of $MKNF_{WF}$ is based on XSB Prolog (xsb.sourceforge.net) for two reasons. First, XSB’s tabling engine evaluates rules according to WFS, and ensures rule termination for programs and goals with the bounded term-size property. Second, the implementation uses the prover from XSB’s ontology management system, the Coherent Description Framework (CDF) (Swift and Warren 2003).

CDF has been used in numerous commercial projects, and was originally developed as a proprietary tool by the company XSB, Inc. Since 2003, CDF has been used to support extraction of information about aircraft parts from free-text data fields, about medical supplies and electronic parts from web-sites and electronic catalogs, and about the specifics of mechanical parts from scanned technical drawings. Also, CDF is used to maintain screen models for graphical user interfaces that are driven by XSB and its graphics package, XJ (www.xsb.com/xj.aspx). We discuss features of CDF that are relevant to the implementation described in Section 4.

Type-0 and Type-1 Ontologies All classes in CDF are represented by terms of the form $cid$(Identifier, Namespace), instances by terms of the form $oid$(Identifier, Namespace), and relations by terms of the form $rid$(Identifier, Namespace), where Identifier and Namespace can themselves be any ground Prolog term.

Commercial use has driven CDF to support efficient query answering from Prolog for very large knowledge bases. A key to this is that 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

---

5 Intuitively, a program $P$ and goal $G$ have the bounded term-size property if there is a finite number $n$ such that all subgoals and answers created in the evaluation of the goal $G$ to $P$ have a size less than $n$.

6 Most of CDF is open-source, including all features used in this paper. CDF is distributed as a package in XSB’s standard release, and full details can be found in its accompanying manual.
A Goal-Directed Implementation of Query Answering for Hybrid MKNF

Fig. 1. Some DL Axioms and their Type-0 Counterparts

than a description logic, and do not add any complexity to query evaluation beyond that of WFS. Support for query answering motivates the representation of Type-0 ontologies. The predicate isa/2 is used to state inclusion: whether the inclusion is a subclass, element of, or subrelation depends on the type of the term (not all combinations of types of terms are allowed in a CDF program). Relational atoms in CDF have the form:

- hasAttr(Term1, Rel1, Term2) which has the meaning Term1 ⊑∃Rel1.Term2;
- allAttr(Term1, Rel1, Term2) with the meaning Term1 ⊑∀Rel1.Term2;
- along with other forms that designate cardinality constraints on relations.

Figure 1 presents some DL Axioms and their Type-0 counterparts, where namespace information has been omitted for readability. The fact that Type-0 ontologies cannot express negation is crucial to their ability to be directly queried. In other words, isa/2, hasAttr/3 and other atoms can be called as Prolog goals, with any instantiation pattern for the call. Query answering for Type-0 goals checks inheritance hierarchies and does not rely on unification alone. Type-0 ontologies use tabling to implement inheritance and use tabled negation so that only the most specific attribute types for a hasAttr/3 or other query are returned to a user.

Besides the constructs of Type-0 ontologies, Type-1 ontologies further allow atoms of the form

necessCond(Term1, CE)

where CE can be any ALCQ class expression over CDF terms. For instance, the axiom

Woman ⊑ Person ⊓ ¬Man

would be represented by the atom

necessCond(cid(woman),(cid(person),neg(cid(man))))

where the comma represents conjunction, as in Prolog. 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 $Term \subseteq ClassExpr$ is provable in the current state of the ontology. Type-1 ontologies deduce entailment using a tableau prover written in Prolog.

**System Features of CDF** Regardless of the type of the ontology, atoms such as $isa/2$, $hasAttr/2$, etc. can be defined extensionally via Prolog atoms, or intensionally via Prolog rules. For instance, evaluation of the goal

$$\text{hasAttr(Class1,Rel,Class2)}$$

would directly check extensional Prolog facts through a subgoal

$$\text{hasAttr_ext(Class1,Rel,Class2)}$$

and would also check intensional rules through a subgoal

$$\text{hasAttr_int(Class1,Rel,Class2)}$$

Intensional definitions are used so that atoms can be lazily defined by querying a database or analyzing a graphical model: their semantics is outside that of CDF. In fact, using combinations of rules and facts, Type-0 ontologies are commonly used comprising tens of thousands of classes and relations, and tens of millions of statements about individuals. At the same time, intensional definitions in a Type-1 ontology provide a basis for the tableau prover to call rules, as is required to support the interdependencies of $MKNF_{WFS}$, and will be further discussed in Section 4.2.

Despite their restrictions, the vast majority of knowledge used by XSB, Inc. is maintained in Type-0 ontologies. Although Type-0 ontologies have supported numerous commercial projects, their limitations of course preclude the full use of information in ontologies. Support of a uniform querying mechanism for individuals in $MKNF_{WFS}$ as described below is intended as a means to allow commercial projects to use a more powerful form of knowledge representation.

**Related Work**

CDF was originally developed in 2002-2003, and Type-0 ontologies were envisioned as a means to represent object-oriented knowledge in Prolog. Unlike Flora-2 (Yang et al. 2003) CDF was intended to be a Prolog library, and its inheritance was designed to be entirely monotonic for compatibility with description logics. Type-1 ontologies were originally evaluated through a translation into ASP (Swift 2004): this approach pre-dated that of KAON2 (Motik 2006) and was abandoned due to the difficulty of dynamically pruning search in ASP; afterwards the current tableau approach was developed which attempts to examine as small a portion of the ontology as possible when proving entailment (cf. e.g. [Horrocks and Patel-Schneider 1999] for a discussion of how search may be pruned in tableau provers for ontologies). CDF’s approach may be distinguished from [Lukacsy et al. 2008], which also combines ontological deduction with Prolog. Type-0 ontologies rely on WFS reasoning and so achieve good scalability under a weak semantics; theorem proving for Type-1 ontologies is used only when needed; [Lukacsy et al. 2008] takes a more uniform approach to deduction which relies on WAM-level extensions for
4 Goal-Driven MKNF Implementation

In Section 2 we presented a bottom-up computation that constructs the complete well-founded model for a given hybrid knowledge base. However, in practical cases, especially when considering the context of the Semantic Web, this is not what is intended. In fact, it would make little sense to compute the whole model of anything that is related to the World Wide Web. Instead, one would like to query the knowledge base for a given predicate (or propositional atom) and determine its truth value. As an illustration, recall Example 1 where we wanted to know if a given shipment should be inspected or not when it arrived, or the case study of [Patel et al. 2007] where one may want to know whether a given patient matches the criteria for a given trial. Deriving all the consequences of a knowledge base to answer a query about a given shipment or patient would be impractical.

In this section we describe the algorithms and the design of CDF-Rules, a goal-driven implementation for Hybrid MKNF Knowledge Bases under the Well-Founded Semantics that minimizes the computation to the set of individuals that are relevant to a query. CDF-Rules makes use of XSB's tabled SLG Resolution [Chen and Warren 1996] for the evaluation of a query, together with tableaux mechanisms supported by CDF theorem prover to check entailment on the ontology. CDF-Rules is tuned for Type-1 ontologies, and thereby is also compatible with Type-0 ontologies. For the description of the solution, we assume that the reader has a general knowledge of tabled logic programs (cf. e.g. (?)).

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 through SLG resolution, 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 $\varphi$ w.r.t. an ontology $\mathcal{O}$ by trying to construct a common model for $\neg \varphi$ and $\mathcal{O}$, sometimes called a completion graph (cf. e.g. [Schmidt-Strauss and Smolka 1990]). If such a model can not be constructed, $\mathcal{O} \models \varphi$; otherwise $\mathcal{O}$ does not entail $\varphi$. Similar to other description logic provers, the CDF theorem prover attempts to traverse as little of an ontology as possible when proving $\varphi$. 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 additional individuals only as necessary.

For our purposes, given the particular interdependence between the rules and the ontology in $\text{MKNF}_{WFS}$, 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 entailed by the ontology. Thus, a query to a DL-atom \( p(o) \), iteratively computes a (sub-)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 3**
Consider the following KB (with the program on the left and the ontology on the right\(^7\)) and the query \( \text{third}(X) \):

\[
\begin{align*}
\text{K third}(X) & \leftarrow p(X), \text{K second}(X), \\
\text{K first}(\text{callback}). & \quad \text{First} \sqsubseteq \text{Second} \\
p(\text{callback}). &
\end{align*}
\]

The query resolves against the rule for \( \text{third}(X) \), leading to the goals \( p(X) \) and \( \text{K second}(X) \). 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 \( X = \text{callback} \). Accordingly, the next subgoal is \( \text{K second}(\text{callback}) \) which depends on the DL-atom \( \text{second}(\text{callback}) \), corresponding in the ontology to the proposition \( \text{Second} \). At this point, the computation calls the CDF theorem prover which starts to derive a model for all the properties of the individual \( \text{callback} \). Yet in this computation, the proposition \( \text{Second} \) itself depends on a predicate defined in the rules – \( \text{First} \). It can thus be seen that the evaluation of the query \( \text{third}(\text{callback}) \) must be done iteratively – the (instantiated) goal \( \text{third}(\text{callback}) \) should suspend (using tabling) until \( \text{second}(\text{callback}) \) is resolved. Furthermore, \( \text{second}(\text{callback}) \) needs first to prove \( \text{first}(\text{callback}) \) from the rules. In general, goals to DL-atoms may need to suspend in order to compute an iterated fixed point, after which they may either succeed or fail.

To formalize the actions in Example 3 on the special case of definite programs, we start by considering the computation for all individuals (i.e. temporarily disregarding the relevance of individuals, as discussed above).

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

---

\(^7\) To simplify reading, for rules we omit the \( K \) 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 \( K \) and \( \text{not} \) are replaced by (meta-)predicates \( \text{known}/1 \) and \( \text{dlnot}/1 \), respectively (see Section 4.2).
The model is obtained as the union of the least fixed point of the sequences:

\[
D_0 = \text{Tableau}(O) \\
D_1 = \text{Tableau}(O \cup R_0) \\
\vdots \\
D_n = \text{Tableau}(O \cup R_{n-1})
\]

Definition 8 resembles Definition 4 of the operator $T_K$ in Section 2. In fact, the $R_i$ sequence is similar to the $R_K(S)$ operator which collects new conclusions from the rules, whereas the $D_i$ sequence is similar to the $D_K(S)$ operator which collects new conclusions from the ontology. Rather than starting with the empty set of conclusions from the rules, as is the case for $T_K$ of Definition 4, here the $R_i$ sequence starts with all conclusions that can be drawn from the program alone. Given these minor differences, taking the function $\text{Tableau}(O)$ as correct w.r.t. the consequence relation of the description logic in use, and taking the function $\text{SLG}(P)$ as correct w.r.t. the least model semantics of definite logic programs, it is easy to see that if $I$ is the set of all individuals in the knowledge base then the union of the $D_n$ and $R_n$ at the fixed point exactly coincides with the least fixed point of $T_K$. Furthermore, as long as the program $P$ respects DL-safety, MKNF rules are lazily grounded with respect to the set of individuals $I$. In fact, given a DL-safe set of MKNF rules, and a set of queries grounded with the individuals in $I$, the evaluation of the queries results in a complete grounding of the rules, and so the fixed point is guaranteed in a finite number of steps.

Definition 8 captures certain aspects of how the rules and the ontology use each other to derive new knowledge in CDF-Rules, via an alternating computation between the rules and the ontology. However it does not capture cases in which the relevant set of individuals changes (i.e. it does not deal with changes in the set $I$), or the presence of default negation in rule bodies. With regard to relevant individuals, since it is possible both to define n-ary predicates in rules and roles in the ontology, a query may depend on 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 else about these individuals, i.e., when both the set of individuals and the classes and roles of those individuals have reached a fixed point. The details of this iterative increase in the set of considered individuals can be found in the algorithm of Figure 2, which also addresses default negation and its interplay with first-order negation.

The following example considers the presence of default negation in rule bodies.

**Example 4**

\[8\] Since $T_K$ is monotonic, it is clear that starting the iteration of this operator with all conclusions drawn from the rules alone would yield exactly the same (least) fixed point.
Consider the following knowledge base:

\[
\begin{align*}
K &\text{fourth}(X) \leftarrow p(X), \text{not} \ third(X). & \text{Fourth} \sqsubseteq \text{Fifth} \\
K &\text{third}(X) \leftarrow p(X), \text{K} \ second(X). & \text{First} \sqsubseteq \text{Second} \\
K &\text{first}(\text{callback}). \\
p(\text{callback}).
\end{align*}
\]

In this example a predicate \text{fourth}(X) is defined as the default negation of \text{third}(X). Since \text{fourth}(X) is defined in the rules, the negation is closed world, that is, \text{fourth}(X) should only succeed if it is not possible to prove \text{third}(X).

Consequently, if we employed SLG resolution blindly, an iteration where the truth of \text{second}(\text{callback}) had not been made available to the rules from the ontology might mistakenly fail the derivation of \text{third}(\text{callback}) and so succeed \text{fourth}(\text{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 \text{fourth}(\text{callback}) was true, it would mistakenly derive \text{Fifth}.

Example 4 illustrates the need to treat default negation carefully, as the truth of default literals requires re-evaluation when new knowledge is inferred. Recall the manner in which the operators \Gamma_K and \Gamma'_K of Definition 6 address the problem of closed-world negation. Roughly, one step in \Gamma_K (or \Gamma'_K) is defined as the application of \text{T}_K until reaching a fixed point. Applying \Gamma'_K followed by \Gamma_K is a monotonic operation and thus is guaranteed to have a least fixed point. In each dual application of \Gamma_K and \Gamma'_K two different models follow – a monotonically increasing model of true atoms (i.e. true predicates and propositions), and a monotonically decreasing model of non-false atoms.

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 \text{T}_K; and an outer fixed point for the evaluation of \text{not}s, corresponding to \Gamma_K (and \Gamma'_K). In the outer iteration, the evaluation of closed-world negation is made by a reference to the previous model obtained by \Gamma_K. Thus in CDF-Rules, \text{not}(A) succeeds if, in the previous outer iteration, A was not proven.

Example 5
To illustrate the need to apply two fixed points, consider the knowledge base below and the query \text{c}(X):

\[
\begin{align*}
K &\text{c}(X) \leftarrow p(X), \text{K} \ a(X), \text{not} \ b(X). & A \sqsubseteq B \\
K &a(\text{object}). \\
p(\text{object}).
\end{align*}
\]

When evaluating the query \text{c}(X), \text{X} is first bound to \text{object} by \text{p}, and then the nested iteration begins. The inner iteration follows the steps of Definition 8 and since these operators are defined only for definite rules, each negative body literal in a rule such as \text{p}(X) is evaluated according to its value in the previous outer fixed point, or is simply evaluated as \text{true} in the first outer iteration. (As we will see, this is done lazily by CDF-Rules). The first stage of the inner iteration computes \(R_0 = \{a(\text{object}), p(\text{object}), c(\text{object})\}\) (Definition 8) via the rules; and computes via
the ontology \( D_0 = \emptyset \), as \( \mathcal{O} \not\models A \). In the second inner stage the rules achieve the same fixed point as in the first, so \( R_1 = R_0 \), but the ontology derives \( \text{object} : B \) in \( D_1 \). After sharing this knowledge, there is nothing else to infer by either components, and we achieve the first inner fixed point with:

\[
T_1 = \{a(\text{object}), b(\text{object}), c(\text{object}), p(\text{object})\}
\]

So now, the second outer iteration starts the computation of the inner iteration again and, in this iteration, negative literals in the rules are evaluated w.r.t. \( T_1 \). As a consequence, \( c(\text{object}) \) fails, since \( b(\text{object}) \in T_1 \). The fixed point of the second inner iteration contains \( p(\text{object}) \), \( a(\text{object}) \) and \( b(\text{object}) \), which is in fact the correct model for \( \text{object} \). Afterwards, a final outer iteration is needed to to determine that an outer fixed point has in fact been reached. Since \( c(\text{object}) \) is in the model of the final iteration, the query \( c(X) \) succeeds for \( X = \text{object} \).

The procedure to compute a lazily invoked iterative fixed point over a DL-safe MKNF Hybrid Knowledge Base is summarized in Figure 2 using predicates that are described in detail in Section 4.2. In each inner iteration, the tabled predicate \( \text{known}/3 \) is used to derive knowledge from the rules component, while \( \text{allModelsEntails}/3 \) infers knowledge from the ontology via tableau proofs. Within rules evaluated by \( \text{known}/3 \), the default negation of a DL-atom \( A \) is obtained by the predicate \( \text{dlnot}(A) \), which succeeds if \( A \) was not proven in the last outer iteration. The predicates \( \text{definedClass}/2 \) and \( \text{definedRole}/3 \) are used to obtain the relevant classes and roles defined for a given individual. We assume that these predicates are defined explicitly by the compiler or programmer, but they can also be inferred via the DL-safe restriction \(^9\). Regardless of whether inference is used, whenever a role is encountered for an individual, a check is made to determine whether the related individual \( \text{Individual}_1 \) is already in the list of individuals in the fixed point, and \( \text{Individual}_1 \) is added if not.

In order to compute \( \text{MKNF}_{WFS} \) the algorithm shown in Figure 2 must create two different sets (cf. Definition 7): a credulous set, \( TU \), containing the atoms that are true or undefined corresponding to application of the operator \( \Gamma'_K \); and a skeptical set, \( T \), of the atoms that are true corresponding to \( \Gamma_K \) (Definition 6). The iterations in Figure 2 capture the construction of these sets in the following manner. The first iteration of \( \text{OutIter} \) in \( \text{known}(\text{Term}, \text{OutIter}, \text{InIter}) \) (where \( \text{OutIter} = 0 \)) corresponds to the first step of computation of the set \( TU \), whereas the second iteration corresponds to the second step of computation of the set \( T \). As a consequence, iterations where \( \text{OutIter} \) is indexed with an even number are monotonically decreasing \( TU \) iterations, while iterations indexed with an odd number are monotonically increasing \( T \) iterations. By making use of this property, the algorithm of Figure 2 ensures that the fixed point will only be achieved in \( TU \) iterations. This way, \( \text{Query} \) is true if \( \text{known}(\text{Query}, \text{OuterFinal} - 1, \text{InnerFinal}) \) holds. If this is not

\(^9\) Because of DL-safety, every DL-rule must contain a positive literal that is only defined in the rules. Such a literal 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.
The Top-Level Algorithm: ComputeFixedPoint(Query)

**Input:** A query Query to a DL-Atom

**Output:** Value of the input query in MKNF_WFS

1. addIndividuals(Query, IndividualList);
2. foreach Individual in IndividualList do
3.   OutIter, InIter = 0;
4.   S = S₁ = {};
5.   P = P₁ = {};
6.   repeat
7.     P = P₁;
8.     repeat
9.       S = S₁;
10.      foreach Class in definedClass(Individual, Class) do
11.        Term = Class(Individual);
12.        S₁ = S₁ ∪ known(Term, OutIter, InIter);
13.        S₁ = S₁ ∪ allModelsEntails(Term, OutIter, InIter);
14.        S₁ = S₁ ∪ allModelsEntails(neg(Term), OutIter, InIter);
15.      endforeach
16.      endforeach
17.      InIter++;
18.    until S = S₁;
19.    P = S;
20.    OutIter++;
21.  endforeach
22.  until OutIter is even (a TU evaluation) and P = P₁;
23. if known(Query, OuterFinal₁, InnerFinal₁) then
24.   return true
25. else
26.   if known(Query, OuterFinal₁, InnerₐFinal₁) then
27.     return undefined
28.   else
29.     return false
30.   end
31. end

Fig. 2. The Top-Level Algorithm: ComputeFixedPoint(Query)

the case, then Query is undefined if known(Query, OuterFinal₁, InnerFinal₁) holds, and Query is false otherwise.

### 4.2 Implementing MKNF\textsubscript{WFS} Components

We now describe the various predicates in the algorithm of Figure 2, including
the manner in which the rule and ontology components exchange knowledge, and how the fixed point is checked.

4.2.1 Rules Component

As mentioned, rules are transformed to use known/1 corresponding to $K$ and $\text{dlnot}/1$ corresponding to not. As shown in Figure 3, given the goal known($A$) with $A = p(O)$, the code first calls computeFixedPoint($p(O)$) to perform the fixed point computation for the object instance $O$. As in Figure 2, computeFixedPoint($p(O)$) calls the lower-level known/3 and $\text{dlnot}/3$ to determine the truth of literals during the fixed point computation. Once the fixed point has been reached, known/1 uses get_object_iter($p(O)$,Outer,Inner) to obtain the final iteration indices for $O$ from a global store, and calls known/3 again to determine the final truth value of $p(O)$. Note that known/3 is always called with the iteration indices (arguments 2 and 3) bound, so that they are always contained in the table entries. Thus, the post-fixed point call to known/3 simply checks the table, and is not computationally expensive.

Within a given iteration, $p(O)$ may be known in one of two ways. Either it can be directly derived from the rules; or $O \in P$ (i.e. $o:P$) may have been entailed by the ontology in the previous inner iteration step, as determined by the call allModelsEntails($p(O)$,Outer,PrevIter). In either case, care must be taken so that if $\neg A$ holds, then not $A$ holds as well. In the formalism of Definition 3, this is guaranteed in two steps. First, an axiom $\neg P \sqsubseteq NP$ is added for each DL-atom that occurs in the head of a rule; in addition, the literal not NP is added the body of each rule with head $P$: this rewrite is used by the $\Gamma'_K$ operator to produce the TU set. Accordingly in Figure 3, when we try to derive known($A$,OutIter,InIter) and OutIter is even (i.e. corresponding to a TU step via $\Gamma'_K$ in Definition 7) we check whether the ontology derived $\neg A$ in the previous inner iteration by the call no_prev_neg($A$,Outer,PrevIter). If $\neg A$ was derived, then no_prev_neg/3 fails via the call to $\text{tnot}/1$, (which is XSB’s operator for tabled negation), and the top-level goal also fails.

On the other hand, the predicate $\text{dlnot}(A)$ which uses closed world assumption, succeeds if $A$ fails (Figure 4). As discussed in Example 5, the evaluation of $\text{dlnot}/2$ must take into account the result of the previous outer iteration. Accordingly, in Figure 4, the call $\text{dlnot}(A)$ with $A = p(O)$ gets the current outer iteration for $O$, and immediately calls $\text{dlnot}/2$. If the outer iteration index is greater than 1, the second clause of $\text{dlnot}/2$ simply finds the index of the (inner) fixed point of the previous outer iteration, and determines whether $A$ was true in that fixed point. Since the call to known/3 in $\text{tnot}/1$ is tabled, $\text{dlnot}/[1,2]$ do not need to be tabled themselves. As described before, outer iterations alternately represent iterations of $T$ and TU sets of Definition 7, where $T$ sets are monotonically increasing while TU sets are monotonically decreasing. To assure that the first TU set is the largest set ($\text{KA}(K^*)$) following Definition 6, we compel all calls to $\text{dlnot}/1$ to succeed in the first outer iteration, as represented by the first clause of $\text{dlnot}/2$. 
known(A):-
    computeFixedPoint(A),
    get_object_iter(A,OutIter,InIter),
    known(A,OutIter,InIter).
:- table known/3.
known(A,OutIter,InIter):-
    PrevIter is InIter - 1,
    ( call(A),
    ;
    InIter > 0, allModelsEntails(A,OutIter,PrevIter) ),
    ( OutIter mod 2 =:= 1 ->
    true
    ;
    no_prev_neg(A,OutIter,PrevIter) ).
/* Enforce coherence of default negation with first-order negation */
no_prev_neg(_A,_OutIter,PrevIter) :-
    PrevIter < 0,!.
no_prev_neg(A,OutIter,PrevIter) :-
    tnot(allModelsEntails(neg(A),OutIter,PrevIter)).

Fig. 3. Prolog Implementation of $K$ for Class Properties

dlnot(A):-
    computeFixedPoint(A),
    get_object_iter(A,OutIter,_InIter),
    dlnot(A,OutIter).
/* In first iteration, ensure that $TU = KA(K^*)$ */
dlnot(_A,0) :- !.
/* In subsequent iterations, check previous outer iteration */
dlnot(A,OutIter):-
    PrevIter is OutIter - 1,
    get_final_iter(A,PrevIter,FinIter),
    tnot(known(A,PrevIter,FinIter)).

Fig. 4. Prolog Implementation of $\text{not}$ for Class Properties

4.2.2 Ontology Component

The tabled predicate allModelsEntails/3 provides the interface to CDF’s tableau theorem prover (Figure 5). It is called with an atom or its classical negation, and with the its iteration indices bound. Although the iterations are not used in the body
of allModelsEntails/3, representing the iteration information in the head ensures its availability in table entries for use by known/3. allModelsEntails/3 first converts the atomic form of a proposition to one used by CDF. That is, it translates a 1-ary DL-atom representing an individual’s class membership to the CDF predicate isa/2, and 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. As is usual with tableau provers, entailment of a formula \( \varphi \) by an ontology \( O \) is shown if the classical negation of \( \varphi \) is inconsistent with \( O \). Thus, rec_allModelsEntails/2 immediately fails if the classical negation of \( \varphi \) is consistent with \( O \) in the present iteration; otherwise, \( \varphi \) is entailed.

The tableau prover, called by rec_allModelsEntails/2, 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. As discussed in Section 3, the architecture of a CDF instance can be divided into two parts – extensional facts and intensional rules. Extensional facts define CDF classes and roles as simple Prolog facts; intensional rules allow classes and roles to be defined by Prolog rules that are outside of the MKNF semantics. In our case, the intensional rules support a programming trick to check rule results from a previous iteration. As shown in Figure 6, they convert the CDF form of an ontology axiom into a 1-ary or 2-ary predicate, and then 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 using definedClass/3 or definedRole/4 before calling lastKnown/1.

4.2.3 Usage

An MKNF Hybrid Knowledge base is defined over a XSB-Prolog knowledge base together with an ontology specified over CDF. In CDF-Rules such a knowledge base is written into two files as follows:

- rules.P – containing the set of MKNF rules and facts. A rule is defined as standard Prolog rules as follows:
isa_int(oid(Obj,NS),cid(Class,NS1)) :-
ground(Obj), ground(Class), !,
Call =.. [Class, Obj], /* Call = Class(Obj) */
lastKnown(Call).
/* Find all possible classes for Obj if called with superclass argument uninstantiated */
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], /* Call = Role(Obj1, Obj2) */
lastKnown(Call).
/* Find all possible rules for Obj if called with role argument uninstantiated */
hasAttr_int(oid(Obj1,NS), rid(Role, NS1), oid(Obj2, NS2)) :-
ground(Obj1), ground(Obj2), var(Role), !,
definedRole(Call, Role, Obj1, Obj2),
lastKnown(Call).

Fig. 6. Callbacks from the ontology component to the rules component

Head :- A_1, ..., A_k, known(B_1), ..., known(B_n), dlnot(C_1), ..., dlnot(C_m).

where k, n, m ≥ 0, and the A_i are all non-DL predicates (i.e. predicates that are not defined in the ontology), and the B_i and C_i are predicates that can be both defined in the rules and in the ontology. If k = n = m = 0 then the rule is a fact, and it is written as usual in Prolog, omitting the ‘:-’ operator.

Note that the transformation to include the negation \( \neg Head \) in the body of a rule for \( Head \) as specified in Definition 3 is not needed: such a check is done by the call to \texttt{no\_prev\_neg/3} in \texttt{known/3}.

To guarantee correctness, each rule must respect DL-safety. However, in the current implementation it is the programmer’s responsibility to check for this condition. The current implementation also does not check that the A_i predicates (i.e. the ones not under \texttt{known/1} or \texttt{dlnot/1}) are not defined in the ontology. If a programmer opts to not precede the predicate by \texttt{known/1} or \texttt{dlnot/1}, any definition for the predicate in the ontology is simply ignored.

- \texttt{cdf\_extensional.P} – comprising ordinary ontology facts and concepts defined over the CDF syntax.
- \texttt{cdf\_intensional.P} – containing predicates allowing the ontology to access information in the rules as in Figure 6. In addition, the file may contain other intensional rules to lazily access information from a database, off of the semantic web, or from other sources external to CDF-Rules.

Example 6
The knowledge base of Example 1 can be easily coded in CDF-Rules as:

```prolog
% rules
inspect(X) :- hasShipment(X,C), dlnot(safeCountry(C)).

% cdf_extensional
isa_ext(cid(scandinavianCtr,ont),cid(safeCountry,ont)).
isa_ext(cid(scandinavianCtr,ont),cid(EuropeanCtr,ont)).
isa_ext(oid(norway,ont),cid(scandinavianCtr,ont)).
necesscond_ext(cid(DiplomaticShipt,ont),neg(cid(inspect,ont))).
```

Note that the ontological portion is Type-1, due to the use of necessCond/2.

### 4.2.4 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 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.

We do not present here a formal proof of soundness and completeness for our algorithm, since this would require the full presentation of the formal derivation procedures on which both XSB-Prolog and CDF implementations rely. However, we have given an informal argument along with our description by referring to complementarity between the implementation and the bottom-up definition of $\text{MKNF}_{\text{WFS}}$. In particular, there is a close correspondence between the inner fixed points of our computation represented in Definition 8 and the $R_K$, $D_K$ and $T_K$ operators of Definition 4; a correspondence between the actions of known/3 in Figure 3 and the transformation of Definition 5 to ensure the coherency between classical and default negation; and also a correspondence between our outer fixed points and the operators $\Gamma_K / \Gamma_K'$. As a result, one can view our goal-driven implementation as an optimization of the bottom-up approach where the computation is limited to the set of relevant objects, and where the evaluation of positive predicates and the handling of iterations is performed by the use of SLG resolution.

Further 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.
In this paper we have 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 (Knorr et al. 2008) and, as such, is also sound w.r.t. the semantics defined in (Motik and Rosati 2007) 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 (Knorr et al. 2008), 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 (?). 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 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. In fact, one of the reasons that highly influenced the choice of CDF as the parameter ontology logic in our query-driven implementation for Hybrid MKNF knowledge bases, was the very existence of an implementation of CDF relying on tabling, that could be coupled together with the tabling we needed for MKNF WFS. But the algorithms presented here do not rely on particularities of CDF, and we believe that, for other choices of parameter logics, implementations could be made in a way similar to the one described in this paper. Of course, such an implementation would require first an implementation in XSB-Prolog of a prover for the other description logic of choice, providing at least a predicate allModelsEntails/3 with the meaning as described above.

Though our choice for the implementation was the Well-Founded Semantics for Hybrid MKNF knowledge bases, MKNF WFS, (Knorr et al. 2008), there were other formalisms concerned with combining ontologies with WFS rules (Eiter et al. 2004b; Drabent and Maluszynski 2007). The approach of (Eiter et al. 2004b) combines ontologies and rules in a modular way, i.e. keeps both parts and their semantics separate, thus having similarities with MKNF WFS. The interface for this approach is done by the dlv-hex system (Schindlauer 2006). Though with identical data complexity to MKNF WFS for a tractable DL, it has a less strong integration, having limitations in the way the ontology can call back program atoms (see (Eiter et al. 2004b) for details). Hybrid programs of (Drabent and Maluszynski 2007) are even more restrictive: this formalism only allows the transfer of information from the ontology to the rules and not the other way around. Moreover, the semantics of this approach differs from MKNF (both the one of (Motik and Rosati 2007) and MKNF WFS) and also (Eiter et al. 2004b) in that if an ontology expresses $B_1 \lor B_2$ then the semantics in (Drabent and Maluszynski 2007) derives $p$ from rules $p \leftarrow$.

10 The proof of tractability of the implementation of CDF-Rules with CDF Type-0 ontologies is beyond the scope of this paper.
$B_1$ and $p \leftarrow B_2$, $p$ while MKNF and (Eiter et al. 2004b) do not. For further comparisons of MKNF with other proposals, including those not based on WFS rules, see Motik and Rosati 2007, Knorr et al. 2008, and for a survey on other proposals for combining rules and ontologies see (Hitzler and Parsia 2009).

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 towards 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 (Yang et al. 2003) and Silk (Grosof 2009) are both based on XSB, CDF-Rules also forms a basis for experimenting with $MKNF_{WFS}$ on these systems.

References


