Query Answering for OWL-DL with Rules

Boris Motik[1], Ulrike Sattler[2], and Rudi Studer[1]

[1] FZI Research Center for Information Technologies, Univ. of Karlsruhe, Germany
{motik,studer}@fzi.de
[2] Department of Computer Science, Univ. of Manchester, UK
sattler@cs.man.ac.uk

Abstract. Both OWL-DL and function-free Horn rules[3] are decidable logics with interesting, yet orthogonal expressive power: from the rules perspective, OWL-DL is restricted to tree-like rules, but provides both existentially and universally quantified variables and full, monotonic negation. From the description logic perspective, rules are restricted to universal quantification, but allow for the interaction of variables in arbitrary ways. Clearly, a combination of OWL-DL and rules is desirable for building Semantic Web ontologies, and several such combinations have already been discussed. However, such a combination might easily lead to the undecidability of interesting reasoning problems. Here, we present a *decidable* such combination which is, to the best of our knowledge, more general than similar decidable combinations proposed so far. Decidability is obtained by restricting rules to so-called *DL-safe* ones, requiring each variable in a rule to occur in a non-DL-atom in the rule body. We show that query answering in such a combined logic is decidable, and we discuss its expressive power by means of a non-trivial example. Finally, we present an algorithm for query answering in $\mathcal{SHIQ}(\mathbf{D})$ extended with DL-safe rules based on the reduction to disjunctive datalog.

1 Introduction

OWL-DL [20] is a W3C recommendation language for ontology representation in the Semantic Web. It is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic (DL), offering a high level of expressivity while still being decidable. For example, $\mathcal{SHOIN}(\mathbf{D})$ provides full negation, disjunction, and (a restricted form of) universal and existential quantification of variables. A related logic, $\mathcal{SHIQ}(\mathbf{D})$ [13, 12], distinguished from $\mathcal{SHOIN}(\mathbf{D})$ mainly by not supporting nominals (or named objects), has been successfully implemented in practical reasoning systems, such as Racer [9] and FaCT [10]. Description logics have been found useful in numerous applications such as information integration [1, ch. 16], software engineering [1, ch. 11], and conceptual modeling [1, ch. 10].

Although OWL-DL is very expressive, it is a *decidable* fragment of first-order logic, and thus cannot express arbitrary axioms: the only axioms it can express are of a certain tree-structure [8]. In contrast, decidable rule-based formalism such as function-free Horn rules do not share this restriction, but lack

[3] Throughout this paper, we use "rules" and "clauses" synonymously, following [11].

some of the expressive power of OWL-DL: they are restricted to universal quantification and lack negation in their basic form. To overcome the limitations of both approaches, OWL-DL was extended with rules in [11], but this extension is undecidable [11]. Intuitively, the undecidability is due to the fact that adding rules to OWL-DL causes the loss of any form of *tree model property*. In a logic with such a property, every satisfiable knowledge base has a model of a certain tree-shaped form. As a consequence, to decide satisfiability (i.e. the existence of a model of a knowledge base), we can search only for such tree-shaped models. For most DLs, it is possible to ensure termination of such a search. To see how rules can destroy this property, consider e.g. the rule $hasAunt(x, y) \leftarrow hasParent(x, z), hasSibling(z, y), Female(y)$, which obviously has only non-tree models.

It is natural to ask what kind of (non-tree) rules can be added to OWL-DL while preserving decidability. This follows a classic line of research in knowledge representation, investigating the trade-off between expressivity and complexity, and providing formalisms with varying expressive power and complexity. It not only provides insight into the causes for the undecidability of the full combination, but also enables a more detailed complexity analysis and, ultimately, the design of "specialized" decision procedures. Applications that do not require the expressive power of the full combination can use such procedures, relying upon known upper time and space bounds required to return a correct answer. Finally, in the last decade, it turned out that many specialized decision procedures are amenable to optimizations, thus achieving surprisingly good performance in practice even for logics with high worst-case complexity [1, ch. 9].

In this paper, we propose a decidable combination of OWL-DL with rules, where decidability is obtained by restricting the rules to so-called *DL-safe* ones. Importantly, we do not restrict the component languages, but only reduce the interface between them. Generalizing the approaches of other decidable combinations of rules and description logics [16, 5], in DL-safe rules, concepts and roles are allowed to occur in both rule bodies and heads as unary, respectively binary predicates in atoms, but each variable of a rule is required to occur in some body literal whose predicate is neither a concept nor a role. We discuss the expressive power and limitations of our approach by means of an example, and show that query answering for it is decidable.

Moreover, we present an algorithm for query answering in the extension of $\mathcal{SHIQ}(\mathbf{D})$ with DL-safe rules which is based on a novel technique for reducing $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases to disjunctive datalog programs [15, 14]. This yields a query answering algorithm which follows the principle of "graceful degradation": the user "pays" only for the features she actually uses. Although a full evaluation is not yet finished, our partial evaluation from [18] is very promising, and we believe that this algorithm can be efficiently realized in practice.

Please note that we are primarily concerned with the semantic and decidability aspects of hybrid reasoning, and not with the infrastructure aspects, such as the syntax or the exchange of rules on the Web. For these issues, we refer the reader to [11] since our approach is fully compatible with the one proposed there.

2 Preliminaries

OWL-DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [11]. Hence, although several XML and RDF syntaxes for OWL-DL exist, in this paper we use the traditional description logic notation since it is more compact. For the correspondence between this notation and various OWL-DL syntaxes, see [11].

$\mathcal{SHOIN}(\mathbf{D})$ supports reasoning with concrete datatypes, such as strings or integers. For example, it is possible to define a minor as a person whose age is less than or equal to 18 in the following way: $\mathit{Minor} \equiv \mathit{Person} \sqcap \exists \mathit{age}.\leq_{18}$. Instead of axiomatizing concrete datatypes in logic, $\mathcal{SHOIN}(\mathbf{D})$ employs an approach similar to [2], where the properties of concrete datatypes are encapsulated in so-called *concrete domains*. A *concrete domain* is a pair $(\triangle_{\mathbf{D}}, \Phi_{\mathbf{D}})$, where $\triangle_{\mathbf{D}}$ is an interpretation domain and $\Phi_{\mathbf{D}}$ is a set of concrete domain predicates with a predefined arity n and an interpretation $d^{\mathbf{D}} \subseteq \triangle_{\mathbf{D}}^n$. An *admissible* concrete domain $\mathbf{D}$ is equipped with a decision procedure for checking satisfiability of finite conjunctions over concrete predicates. Satisfiability checking of admissible concrete domains can successfully be combined with logical reasoning for many description logics [17].

We use a set of concept names N_C, sets of abstract and concrete individuals N_{I_a} and N_{I_c}, respectively, and sets of abstract and concrete role names N_{R_a} and N_{R_c}, respectively. An *abstract role* is an abstract role name or the inverse S^- of an abstract role name S (concrete roles do not have inverses). In the following, we assume that $\mathbf{D}$ is an admissible concrete domain.

An *RBox* $\mathcal{R}$ consists of a finite set of transitivity axioms $\mathsf{Trans}(R)$, and role inclusion axioms of the form $R \sqsubseteq S$ and $T \sqsubseteq U$, where R and S are abstract roles, and T and U are concrete roles. The reflexive-transitive closure of the role inclusion relationship is denoted with $\sqsubseteq^*$. A role not having transitive subroles (w.r.t. $\sqsubseteq^*$, for a full definition see [13]) is called a *simple* role.

The set of $\mathcal{SHOIN}(\mathbf{D})$ *concepts* is defined by the following syntactic rules, where A is an atomic concept, R is an abstract role, S is an abstract simple role, $T_{(i)}$ are concrete roles, d is a concrete domain predicate, a_i and c_i are abstract and concrete individuals, respectively, and n is a non-negative integer:

$$
\begin{aligned}
C &\rightarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n\, S \mid \leq n\, S \mid \{a_1, \ldots, a_n\} \mid \\
&\quad \mid \geq n\, T \mid \leq n\, T \mid \exists T_1, \ldots, T_n.D \mid \forall T_1, \ldots, T_n.D \\
D &\rightarrow d \mid \{c_1, \ldots, c_n\}
\end{aligned}
$$

A *TBox* $\mathcal{T}$ consists of a finite set of concept inclusion axioms $C \sqsubseteq D$, where C and D are concepts; an *ABox* $\mathcal{A}$ consists of a finite set of concept and role assertions and individual (in)equalities $C(a)$, $R(a, b)$, $a \approx b$, and $a \not\approx b$, respectively. A $\mathcal{SHOIN}(\mathbf{D})$ *knowledge base* $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$, and an ABox $\mathcal{A}$.

The $\mathcal{SHIQ}(\mathbf{D})$ description logic is obtained from $\mathcal{SHOIN}(\mathbf{D})$ by disallowing nominal concepts of the form $\{a_1, \ldots, a_n\}$ and $\{c_1, \ldots, c_n\}$, and by allowing qualified number restrictions of the form $\geq n\, S.C$ and $\leq n\, S.C$, for C a $\mathcal{SHIQ}(\mathbf{D})$ concept and S a simple role.

Mapping Concepts to FOL	
$\pi_y(\top, X) = \top$	$\pi_y(\bot, X) = \bot$
$\pi_y(A, X) = A(X)$	$\pi_y(\neg C, X) = \neg \pi_y(C, X)$
$\pi_y(C \sqcap D, X) = \pi_y(C, X) \wedge \pi_y(D, X)$	$\pi_y(C \sqcup D, X) = \pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X) = \forall y : R(X, y) \rightarrow \pi_x(C, y)$	$\pi_y(\exists R.C, X) = \exists y : R(X, y) \wedge \pi_x(C, y)$
$\pi_y(\{a_1 \ldots, a_n\}, X) = X \approx a_1 \vee \ldots \vee X \approx a_n$	
$\pi_y(\leq n\, R.C, X) = \forall y_1, \ldots, y_{n+1} : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \rightarrow \bigvee y_i \approx y_j$	
$\pi_y(\geq n\, R.C, X) = \exists y_1, \ldots, y_n : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \wedge \bigwedge y_i \not\approx y_j$	
$\pi_y(\forall T_1, \ldots, T_m.d, X) = \forall y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \rightarrow d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$	
$\pi_y(\exists T_1, \ldots, T_m.d, X) = \exists y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \wedge d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$	
$\pi_y(\leq n\, T, X) = \forall y_1^{\mathsf{c}}, \ldots, y_{n+1}^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \rightarrow \bigvee y_i^{\mathsf{c}} \approx y_j^{\mathsf{c}}$	
$\pi_y(\geq n\, T, X) = \exists y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \wedge \bigwedge y_i^{\mathsf{c}} \not\approx y_j^{\mathsf{c}}$	
Mapping Axioms to FOL	
$\pi(C(a)) = \pi_y(C, a)$	$\pi(R(a, b)) = R(a, b)$
$\pi(a \approx b)) = a \approx b$	$\pi(a \not\approx b)) = a \not\approx b$
$\pi(C \sqsubseteq D) = \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$	
$\pi(R \sqsubseteq S) = \forall x, y : R(x, y) \rightarrow S(x, y)$	
$\pi(\mathsf{Trans}(R)) = \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$	
Mapping KB to FOL	
$\pi(KB) = \bigwedge_{R \in N_R} \forall x, y : R(x, y) \leftrightarrow R^-(y, x) \wedge \bigwedge_{\alpha \in KB_{\mathcal{R}} \cup KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} \pi(\alpha)$	
where X is a meta variable and is substituted by the actual variable and π_x is defined as π_y by substituting x and x_i for all y and y_i, respectively.	

Table 1. Translation of $\mathcal{SHOIN}(\mathbf{D})$ into FOL

Since the algorithms we present in Section 5 are based on resolution, instead of using a direct model-theoretic semantics to $\mathcal{SHOIN}(\mathbf{D})$ [13], we present an equivalent semantics by translation into multi-sorted first-order logic. To separate the interpretations of the abstract and the concrete domain, we introduce the sorts a and c, and use the notation x^{c} and f^{c} to denote that x and f are of sort c. We translate each atomic concept into a unary predicate of sort a, each n-ary concrete domain predicate into a predicate with arguments of sort c, and each abstract (concrete) role into a binary predicate of sort $\mathsf{a} \times \mathsf{a}$ ($\mathsf{a} \times \mathsf{c}$). The translation operator π is presented in Table 1.

For rules, we use the standard definitions. Let N_P be a set of predicate symbols such that $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. A *term* is either a constant (denoted by a, b, c) or a variable (denoted by x, y, z). An *atom* has the form $P(s_1, \ldots, s_n)$, where P is a predicate symbol and s_i are terms. A *rule* has the form

$$H \leftarrow B_1, \ldots, B_n$$

where H and B_i are atoms; H is called the *rule head*, and the set of all B_i is called the *rule body*. A *program* P is a finite set of rules. For the semantics, we define a rule $H \leftarrow B_1, \ldots, B_n$ to be equivalent to the clause $H \vee \neg B_1 \vee \ldots \vee \neg B_n$. This yields a monotonic formalism compatible with the one from [11].

$Person(Peter)$	Peter is a person.
$Person \sqsubseteq \exists father.Person$	Each person has a father who is a person.
$\exists father.(\exists father.Person) \sqsubseteq Grandchild$	Things having a father of a father who is a person are grandchildren.

Table 2. Example Knowledge Base

3 Reasons for the Undecidability of OWL-DL with Rules

In [11], the following problem was shown to be undecidable: given an OWL-DL knowledge base KB and a program P, is there a common model of $\pi(KB)$ and P, i.e. is KB consistent with P? As a consequence, subsumption and query answering w.r.t. knowledge bases and programs is also undecidable. Investigating this proof and the ones in [16] more closely, we note that the undecidability is caused by the interaction between some very basic features of description logics and rules. In this section, we try to give an intuitive explanation of this result and its consequences.

Consider the simple knowledge base KB from Table 2. It is not too difficult to see that this knowledge base implies the existence of an infinite chain of fathers: since $Peter$ must have a father, there is some x_1 who is a $Person$. In turn, x_1 must have some father x_2, which must be a $Person$, and so on. An infinite model with such a chain is shown in Figure 1, upper part a). Observe that $Peter$ is a grandchild, since he has a father of a father.

Let us now check whether $KB \models Grandchild(Jane)$; this is the case if and only if $KB \cup \{\neg Grandchild(Jane)\}$ is unsatisfiable, i.e. if it does not have a model. We can check this by trying to build such a model; if we fail, then we conclude that $KB \cup \{\neg Grandchild(Jane)\}$ is unsatisfiable. However, we have a problem: starting from $Peter$, a naïve approach to building a model will expand the chain of Peter's fathers indefinitely, and will therefore not terminate.

This very simple example intuitively shows that we have to be careful if we want to ensure termination of a satisfiability checking algorithm. For many DLs, termination can be ensured without losing correctness because we can restrict our attention to certain "nice" models. For numerous DLs, we can restrict our attention to *tree models*, i.e. to models where the underlying relational structure forms a tree [22]. This is so because every satisfiable knowledge base has such a tree model (to be precise, for some DLs we consider tree-like abstractions of

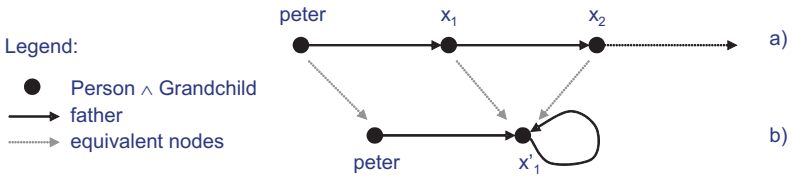

Fig. 1. Two Similar Models

non-tree-like models). Even if such a tree model is infinite, we can *wind* this infinite tree model into a finite one. In our example, since KB does not require each father in the chain to be distinct, the model in Figure 1, lower part b) is the result of this "winding" of a tree into a "nice" model. Due to their regular structure, these "windings" of tree models can be easily constructed in an automated way. To understand why every satisfiable $\mathcal{SHIQ}(\mathbf{D})$ knowledge base has a tree model [13], consider the mapping π in Table 1 more closely (we abstract some technicalities caused by the transitive roles): in all formulae obtained by transforming the result of π into prenex normal form, variables are connected by roles only in a tree-like manner, as shown in the following example:

$$\begin{array}{l} \exists S.(\exists R.C \sqcap \exists R.D) \sqsubseteq Q \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow \\ \forall x : \{[\exists y : S(x,y) \wedge (\exists x : R(y,x) \wedge C(x)) \wedge (\exists x : R(y,x) \wedge D(x))] \rightarrow Q(x)\} \Rightarrow \\ \forall x, x_1, x_2, x_3 : \{S(x,x_1) \wedge R(x_1,x_2) \wedge C(x_2) \wedge R(x_1,x_3) \wedge D(x_3) \rightarrow Q(x)\} \end{array}$$

Let us contrast these observations with the kind of reasoning required for function-free Horn rules. In such rules, all variables are universally quantified, i.e. there are no existentially quantified variables in rule consequents. Hence, we never have to infer the existence of "new" objects. Thus, reasoning algorithms must consider only individuals explicitly introduced in the knowledge base and will never run into the termination problems outlined above. Hence, the rules, such as the one defining $hasAunt(x, y)$ from the introduction, are allowed to enforce arbitrary but finite, non-tree relational models, and not only "nice" models.

Now let us see what happens if we extend, eg. $\mathcal{SHIQ}$, with function-free Horn rules. Then, we combine a logic whose decidability is due to the fact that we can restrict our attention to "nice" models (but with individuals whose existence may be implied by a knowledge base) with the one whose decidability is due to the fact that we can restrict our attention to "known" individuals (but with arbitrary relations between them). Unsurprisingly, this and similar combinations are undecidable [16, 11].

4 DL-safe Rules

As a reaction to the observations in Section 3, in this section we define the formalism of *DL-safe* rules, discuss its benefits and drawbacks, and prove that query answering in $\mathcal{SHOIN}$ with DL-safe rules is decidable.

Definition 1 (DL-safe Rules). *Let KB be a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base, and let N_P be a set of predicate symbols such that $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. A* DL-atom *is an atom of the form $A(s)$, where $A \in N_C$, or of the form $R(s,t)$, where $R \in N_{R_a} \cup N_{R_c}$. A rule r is called* DL-safe *if each variable in r occurs in a non-DL-atom in the rule body. A program P is* DL-safe *if all its rules are DL-safe.*

The semantics of the combined knowledge base *(KB, P) is given by translation into first-order logic as $\pi(KB) \cup P$. The main inference in (KB, P) is* query answering*, i.e. deciding whether $\pi(KB) \cup P \models \alpha$ for a ground atom α.*

Some remarks are in order. Firstly, DL-safety is similar to the safety in datalog. In a safe rule, each variable occurs in a positive atom in the body, and may therefore be bound only to constants explicitly present in the database. Similarly, DL-safety makes sure that each variable is bound only to individuals explicitly introduced in the ABox. For example, if *Person*, *livesAt*, and *worksAt* are concepts and roles from KB, the following rule is not DL-safe:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y)$$

The reason for this is that both variables x and y occur in DL-atoms, but do not occur in a body atom with a predicate outside of KB. This rule can be made DL-safe by adding special non-DL-literals $\mathcal{O}(x)$ and $\mathcal{O}(y)$ to the rule body, and by adding a fact $\mathcal{O}(a)$ for each individual a. In Subsection 4.1 we discuss the consequences that this transformation has on the semantics.

Secondly, DL-safety only allows atomic concepts to occur in a rule. This is not really a restriction: for a complex concept C, one may introduce an atomic concept A_C, add the axiom $C \sqsubseteq A_C$ to the TBox, and use A_C in the rule.

4.1 Expressivity of DL-safe Rules

In our approach, to achieve decidability, we do not restrict the component languages. Rather, we combine full $\mathcal{SHOIN}(\mathbf{D})$ with function-free Horn rules, and thus extend both formalisms. DL-safety only restricts the interchange of consequences between the component languages to those consequences involving individuals explicitly introduced in the ABox.

To illustrate the expressive power of DL-safe rules, we extend the example from Table 2 with the TBox axioms and rules from Table 3. We use a rule to define the only non-DL-predicate *BadChild* as a grandchild which hates some of its siblings (or itself). Notice that this rule involves relations forming a triangle between two siblings and a parent and thus cannot be expressed in a description logic such as $\mathcal{SHOIN}(\mathbf{D})$. Moreover, it is not DL-safe because each variable in the rule does not occur in a non-DL-atom in the rule body.

Now consider the first group of ABox facts. Since *Cain* is a *Person*, as in Section 3 one may infer that *Cain* is a *Grandchild*. Since *Cain* and *Abel* are children of *Adam*, and *Cain hates Abel*, *Cain* is a *BadChild*.

Similarly, *Romulus* has a father who is a father of *Remus*, and *Romulus hates Remus*, so *Romulus* is a *BadChild* as well. We are able to derive this without knowing exactly who the father of *Romulus* is[4].

Consider now the DL-safe rule defining *BadChild′* (assuming that the ABox contains $\mathcal{O}(a)$ for each individual a): since the father of *Cain* and *Abel* is known by name (i.e. *Adam* is in the ABox), the literal $\mathcal{O}(y)$ from the rule for *BadChild′* can be matched to $\mathcal{O}(Adam)$, and we may conclude that *Cain* is a *BadChild′*. In contrast, the father of *Romulus* and *Remus* is not known in the ABox. Hence,

[4] Historically, the father of Romulus and Remus is Mars, but for illustration purposes we assume that the modeler does not know that.

$father \sqsubseteq parent$	Fatherhood is a kind of parenthood.
$BadChild(x) \leftarrow Grandchild(x),$ $parent(x,y), parent(z,y), hates(x,z)$	A bad child is a grandchild who hates one of his siblings.
$BadChild'(x) \leftarrow Grandchild(x),$ $parent(x,y), parent(z,y), hates(x,z),$ $\mathcal{O}(x), \mathcal{O}(y), \mathcal{O}(z)$	DL-safe version of a bad child.
$Person(Cain)$	Cain is a person.
$father(Cain, Adam)$	Cain's father is Adam.
$father(Abel, Adam)$	Abel's father is Adam.
$hates(Cain, Abel)$	Cain hates Abel.
$Person(Romulus)$	Romulus is a person.
$\exists father.\exists father^{-}.\{Remus\}(Romulus)$	Romulus' father is a father of Remus.
$hates(Romulus, Remus)$	Romulus hates Remus.
$Child(x) \leftarrow GoodChild(x), \mathcal{O}(x)$	Good children are children.
$Child(x) \leftarrow BadChild'(x), \mathcal{O}(x)$	Bad children are children.
$(GoodChild \sqcup BadChild')(Oedipus)$	Oedipus is a good or a bad child.
$\mathcal{O}(\alpha)$ for each explicitly named individual α	Enumeration of all ABox individuals.

Table 3. Example with DL-safe Rules

the literal $\mathcal{O}(y)$ from the DL-safe rule cannot be matched to the father's name, so the rule does not derive that *Romulus* is a *BadChild′*.

This may seem confusing. However, DL-safe rules do have a "natural" reading: just append the phrase "where the identity of all objects is known" to the meaning of the rule. For example, the rule defining *BadChild′* can be read as "A *BadChild′* is a *known* grandchild for which we *know* a parent, and who hates one of his *known* siblings".

Combining description logics with DL-safe rules increases the expressivity of both languages. Namely, a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base cannot imply that *Cain* is a *BadChild′* because the "triangle" rule cannot be expressed using $\mathcal{SHOIN}(\mathbf{D})$ constructs. Similarly, a set of function-free Horn rules cannot imply this either: we know that Cain has a grandfather because Cain is a person, but we do not know who he is. Hence, we need the existential quantifier to *infer* the existence of ancestors, and then to infer that *Cain* is a *Grandchild*.

Finally, we would like to point out that it is incorrect to compute all consequences of the description logic component first, and then to apply the rules to the consequences. Consider the *KB* part about *Oedipus*: he is a *GoodChild* or a *BadChild′*, but we do not know exactly which. Either way, one of the rules derives that *Oedipus* is a *Child*, so $(KB, P) \models Child(Oedipus)$. This would not be derived by applying the rules to the consequences of *KB*, since $KB \not\models GoodChild(Oedipus)$ and $KB \not\models BadChild'(Oedipus)$.

4.2 Decidability of Query Answering

We now sketch a proof for the decidability of query answering for $\mathcal{SHOIN}$ extended with DL-safe rules, which is by a non-deterministic reduction of the query answering problem to the satisfiability problem for $\mathcal{SHOIN}$ without rules.

Theorem 1. *For a $\mathcal{SHOIN}$ knowledge base KB and a DL-safe program P, query answering in (KB, P) is decidable.*

Proof. Clearly $(KB, P) \models \alpha$ iff $\pi(KB) \cup P'$ is unsatisfiable, where $P' = P \cup \{\neg\alpha\}$. Let P^g be the set of ground instances of P', i.e. P^g contains all possible ground instantiations of rules in P' with individuals from KB and P.

We now show that $\pi(KB) \cup P'$ is satisfiable iff $\pi(KB) \cup P^g$ is satisfiable. The ($\Rightarrow$) direction is trivial. For the ($\Leftarrow$) direction, let I be a model of $\pi(KB) \cup P^g$. Since $\pi(KB) \cup P^g$ does not contain non-DL-atoms with variables, we may safely assume that the interpretation of each non-DL-predicate contains only tuples of the form $(\alpha_1, \ldots, \alpha_n)$, such that, for each α_i, there is a constant a_i with $a_i^I = \alpha_i$. Let r be a rule from P. Since r is DL-safe, each variable in r occurs in a body non-DL-atom. Hence, for each valuation replacing a variable in r with an individual α, for which there is no such constant a with $a^I = \alpha$, there will be a body atom of r which is false in I, making r true in I. Thus, I is a model of $\pi(KB) \cup P'$.

Satisfiability of $\pi(KB) \cup P^g$ can be decided by case analysis as follows: each model of P^g satisfies at least one literal per rule. Hence, we don't-know non-deterministically choose one literal per clause in P^g and, for L^c the resulting set of literals, we test the satisfiability of $\pi(KB) \cup L^c$. Clearly, $\pi(KB) \cup P^g$ is satisfiable iff there exists a "choice" of L^c such that $\pi(KB) \cup L^c$ is satisfiable.

Next, let $L^c_{DL} \subseteq L^c$ be the set of (ground) literals in L^c involving DL predicates. Clearly, $\pi(KB) \cup L^c$ is unsatisfiable iff either L^c contains a complementary pair of ground literals or $\pi(KB) \cup L^c_{DL}$ is unsatisfiable. The first case can be checked easily, and the second case can be reduced to standard $\mathcal{SHOIN}$ reasoning as follows: L^c_{DL} can be viewed as an ABox, apart from literals of the form $\neg R(a, b)$. However, each such literal can be transformed into an equivalent $\mathcal{SHOIN}$ ABox assertion $(\forall R.\neg\{b\})(a)$. Thus we have reduced query answering to deciding satisfiability of a $\mathcal{SHOIN}$ knowledge base. This problem is decidable because (*i*) transitivity axioms can be eliminated from $\mathcal{SHOIN}$ knowledge bases in the same way as this is done for $\mathcal{SHIQ}$ in [14] and (*ii*) the resulting logic is a syntactic variant of the two variable fragment of first-order logic with counting quantifiers, which is known to be decidable [7]. □

We strongly believe that Theorem 1 also holds for $\mathcal{SHOIN}(\mathbf{D})$: (*i*) the decidability proof of $\mathcal{SHOIN}$ should be easily adaptable to $\mathcal{SHOIN}(\mathbf{D})$, and (*ii*) the same non-deterministic reduction of ground DL-safe rules to sets of ground literals as for $\mathcal{SHOIN}$ is applicable to $\mathcal{SHOIN}(\mathbf{D})$. To work out the details of this proof is part of our future work.

5 Query Answering with DL-safe Rules

In the proof of the Theorem 1, we have presented a decision procedure for query answering in the full combination from Section 4. However, this procedure is likely to be hopelessly inefficient in practice, mainly due to the huge amount

of don't-known non-determinism. Hence, in this section, we describe a practical reasoning algorithm for the following fragment: (*i*) the description logic is $\mathcal{SHIQ}(\mathbf{D})$, and (*ii*) in rules, DL-atoms are restricted to concepts and simple roles. Our algorithm is based on reducing the description logic knowledge base to a positive disjunctive datalog program which entails the same set of ground facts as the original knowledge base. DL-safe rules (with the above restriction to concepts and simple roles) can simply be appended to such a program. For unary coding of numbers and assuming a bound on the arity of predicates in rules, our algorithm runs in deterministic exponential time, which makes it optimal since $\mathcal{SHIQ}$ is ExpTime-complete [21].

The full presentation of the algorithm and a proof of its correctness are technically involved and lengthy. Here, we just provide an overview of the procedure, without going into details. For a complete presentation of the procedure and for the proofs of its correctness, we direct the interested reader to [14, 15].

Our algorithm does not support all of $\mathcal{SHOIN}(\mathbf{D})$ since it does not support nominals: to the best of our knowledge, no decision procedure has yet been implemented for $\mathcal{SHOIN}(\mathbf{D})$. The development of such a decision procedure is part of our ongoing work. Namely, the combination of nominals, inverse roles, and number restriction is known to be difficult to handle, which is confirmed by the increase in complexity from ExpTime to NExpTime [21].

5.1 Reducing $\mathcal{SHIQ}(\mathbf{D})$ to Disjunctive Datalog

Let KB be a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base. The reduction of KB to a disjunctive datalog program $\mathsf{DD}(KB)$ can be computed by an algorithm schematically presented in Figure 2. We next explain each step of the algorithm.

Elimination of Transitivity Axioms. Our core algorithms cannot handle transitivity axioms, basically because in their first-order logic formulation they involve three variables. However, we can eliminate transitivity axioms by encoding KB into an equisatisfiable knowledge base $\Omega(KB)$. Roughly speaking, for each transitive role S, each role $S \sqsubseteq^* R$, and each concept $\forall R.C$ occurring in KB, it is sufficient to add an axiom $\forall R.C \sqsubseteq \forall S.(\forall S.C)$. Intuitively, this axiom propagates all relevant concept constraints through transitive roles. Whereas KB and $\Omega(KB)$ entail the same set of ground facts concerning simple roles, they do not entail the same set of ground facts concerning complex roles. This is the reason for the restriction (*ii*) which allows only simple roles to occur in DL-safe rules.

Translation into Clauses. The next step is to translate $\Omega(KB)$ into clausal first-order logic. We first use π as defined in Table 1 and then transform the result

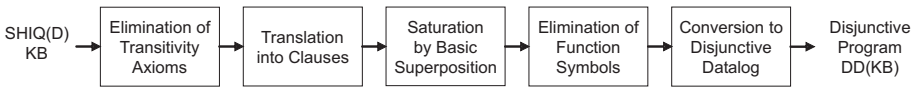

Fig. 2. Algorithm for Reducing $\mathcal{SHIQ}(\mathbf{D})$ to Datalog Programs

$\pi(\Omega(KB))$ into clausal form using *structural transformation* to avoid exponential blow-up [19]. We call the result $\Xi(KB)$.

Saturation by Basic Superposition. We next saturate the RBox and TBox clauses of $\Xi(KB)$ by basic superposition [4] — a clausal calculus optimized for theorem proving with equality. In this key step of the reduction, we compute all non-ground consequences of KB. We can prove that saturation terminates because application of each rule of basic superposition produces a clause with at most one variable and with functional terms of depth at most two. This yields an exponential bound on the number of clauses we can compute, and thus an exponential time complexity bound for our algorithm so far.

Elimination of Function Symbols. Saturation of RBox and TBox of $\Xi(KB)$ computes all non-ground consequences of KB. If we add ABox assertions to this saturated clause set, all "further" inferences by basic superposition will produce only ground clauses. Moreover, the resulting ground clauses contain only ground functional terms of depth one. Hence, it is possible to simulate each functional term $f(a)$ with a new constant a_f. For each function symbol f, we introduce a binary predicate S_f, and for each individual a, we add an assertion $S_f(a, a_f)$. Finally, if a clause contains the term $f(x)$, we replace it with a new variable x_f and add the literal $\neg S_f(x, x_f)$, as in the following example:

$$\neg C(x) \vee D(f(x)) \Rightarrow \neg S_f(x, x_f) \vee \neg C(x) \vee D(x_f)$$

We denote the resulting function-free set of clauses with $\mathsf{FF}(KB)$. In [14], we show that each inference step of basic superposition in $\Xi(KB)$ can be simulated by an inference step in $\mathsf{FF}(KB)$, and vice versa. Hence, KB and $\mathsf{FF}(KB)$ are equisatisfiable.

Conversion to Disjunctive Datalog. Since $\mathsf{FF}(KB)$ does not contain functional terms and all its clauses are safe, we can rewrite each clause into a positive disjunctive rule. We use $\mathsf{DD}(KB)$ for the result of this rewriting.

The following theorem summarizes the properties of our algorithm (we use $\models_c$ for cautions entailment in disjunctive datalog, which coincides on ground facts with first-order entailment for positive datalog programs [6]):

Theorem 2 ([14]). *Let KB be an $\mathcal{SHIQ}(\mathbf{D})$ knowledge base, defined over an admissible concrete domain $\mathbf{D}$, such that satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in deterministic exponential time. Then the following claims hold:*

1. *KB is unsatisfiable if and only if $\mathsf{DD}(KB)$ is unsatisfiable.*
2. *$KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, for α of the form $A(a)$ or $S(a, b)$, A an atomic concept, and S a simple role.*
3. *$KB \models C(a)$ if and only if $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\}) \models_c Q(a)$, for C a non-atomic concept, and Q a new atomic concept.*
4. *Let $|KB|$ be the length of KB with numbers in number restrictions coded in unary. The number of rules in $\mathsf{DD}(KB)$ is at most exponential in $|KB|$, the number of literals in each rule is at most polynomial in $|KB|$, and $\mathsf{DD}(KB)$ can be computed in time exponential in $|KB|$.*

Adding DL-safe Rules. The disjunctive program $\mathsf{DD}(KB)$ can be combined with DL-safe rules by simply appending the rules to the program. The following theorem shows that (KB, P) and $\mathsf{DD}(KB) \cup P$ entail the same sets of ground facts:

Theorem 3 ([14]). *Let KB be a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base and P a DL-safe disjunctive datalog program. Then $(KB, P) \models \alpha$ if and only if $\mathsf{DD}(KB) \cup P \models_c \alpha$, where α is a DL-atom $A(a)$ or $S(a, b)$ for S a simple role, or α is a ground non-DL-atom.*

5.2 Evaluating Queries in Datalog Program

Answering queries in disjunctive datalog programs is computationally more expensive than in non-disjunctive programs [6]. Furthermore, if disjunction is not used in a knowledge base, our algorithm should not introduce a performance penalty. To address that, we have devised an algorithm for evaluating queries in disjunctive programs, which we outline next. For details, please see [14].

Let P be a positive datalog program and let Q be a query predicate not occurring in the body of a rule of P. This is not a limitation since one can always introduce a new predicate A_Q and add the rule $A_Q(\mathbf{x}) \leftarrow Q(\mathbf{x})$. To compute all answers of $\mathbf{a}$ to Q w.r.t. P, i.e. all $\mathbf{a}$ such that $P \models_c Q(\mathbf{a})$, we saturate P by hyperresolution and perform paramodulation inferences between ground clauses to deal with equality. It is well-known that this calculus remains complete if ground literals are totally ordered under an arbitrary ordering $\succ$, and inferences are performed on maximal literals only [3]. This ordering has the useful property that, in each ground disjunction, exactly one literal is maximal. Hence, instead of performing an inference on each literal of a ground fact, it is sufficient to do so on the maximal literal only, which dramatically improves performance.

In [14] we have shown that, if all literals involving the predicate Q are minimal w.r.t. $\succ$, then all ground consequences of P will be derived as *unit* ground clauses in the saturation. To compute all answers to a query, it is sufficient to saturate P only once using an appropriate ordering $\succ$. Thus, our algorithm computes all consequences by a single saturation.

An example of a hyperresolution inference is presented in Figure 3. The maximal literals in the premises are underlined, and only these literals can participate in an inference. When performing hyperresolution, premises are matched to a disjunctive rule exactly as in the non-disjunctive case, the variables in the rule

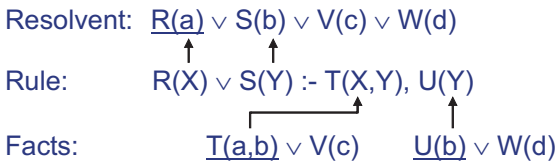

Fig. 3. Hyperresolution Example

head are instantiated, and the remaining literals from the rule body are transferred to the rule head. Observe that, if premises and the rule are not disjunctive, then hyperresolution becomes exactly the least fixpoint operator used to evaluate non-disjunctive datalog programs. The consequences of the least fixpoint operator can be computed in polynomial time, so we get tractable behavior. In this way our algorithm supports the principle of "graceful degradation": the user pays a performance penalty only for features actually used.

6 Related Work

$\mathcal{AL}$-log [5] is a logic which combines a TBox and ABox expressed in the basic description logic $\mathcal{ALC}$ with datalog rules, which may be constrained with unary atoms having $\mathcal{ALC}$ concepts as predicates in the body. Query answering in $\mathcal{AL}$-log is decided by a variant of constrained resolution, combined with a tableaux algorithm for $\mathcal{ALC}$. The combined algorithm is shown to run in single non-deterministic exponential time. The fact that atoms with concept predicates can occur only as constraints in the body makes rules applicable only to explicitly named objects. Our restriction to DL-safe rules has the same effect. However, our approach is more general in the following ways: (*i*) it supports a more expressive description logic, (*ii*) it allows using both concepts and roles in DL-atoms and (*iii*) DL-atoms can be used in rule heads as well. Furthermore, (*iv*) we present a query answering algorithm as an extension of deductive database techniques which runs in deterministic exponential time.

A comprehensive study of the effects of combining datalog rules with description logics is presented in [16]. The logic considered is $\mathcal{ALCNR}$, which, although less expressive than $\mathcal{SHOIN}(\mathbf{D})$ or $\mathcal{SHIQ}(\mathbf{D})$, contains constructors that are characteristic of most DL languages. The results of the study can be summarized as follows: (*i*) answering conjunctive queries over $\mathcal{ALCNR}$ TBoxes is decidable, (*ii*) query answering in a logic obtained by extending $\mathcal{ALCNR}$ with non-recursive datalog rules, where both concepts and roles can occur in rule bodies, is also decidable, as it can be reduced to computing a union of conjunctive query answers, (*iii*) if rules are recursive, query answering becomes undecidable, (*iv*) decidability can be regained by disallowing certain combinations of constructors in the logic, and (*v*) decidability can be regained by requiring rules to be *role-safe*, where at least one variable from each role literal must occur in some non-DL-atom. As in $\mathcal{AL}$-log, query answering is decided using constrained resolution and a modified version of the tableaux calculus. Besides the fact that we treat a more expressive logic, in our approach all variables in a rule must occur in at least one non-DL-atom, but concepts and roles are allowed to occur in rule heads. Hence, when compared to the variant (*v*), our approach is slightly less general in some, and slightly more general in other aspects.

The OWL Rule Language (ORL) [11] combines OWL-DL with rules in which concept and role predicates are allowed to occur in the head and in the body, without any restrictions. As mentioned before, this combination is undecidable but, as pointed out by the authors, (incomplete) reasoning in such a logic can

be performed using general first-order theorem provers. Our approach trades some expressivity for decidability. Furthermore, we provide an optimal query answering algorithm covering a significant portion of OWL-DL.

Finally, [8] describes a natural, decidable intersection of description logic and logic programming. This provides a useful insight into the relationship between these two formalisms, but yields a combination which is less expressive than our approach, since it does not support existential quantifiers, negation, or disjunction in the axiom consequent.

7 Summary and Outlook

We have presented an approach for extending OWL-DL with DL-safe rules. Instead of reducing the component formalisms, we reduce the interface between them. As a consequence, rules apply only to individuals explicitly introduced in the ABox. We have discussed the effects of such a definition on a non-trivial example, which also shows that our approach increases the expressivity of its two components.

Besides a decidability result for $\mathcal{SHOIN}$ with DL-safe rules, we have presented an algorithm for answering queries over $\mathcal{SHIQ}(\mathbf{D})$ extended with DL-safe rules which we believe to be useful in practice. This algorithm transforms a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base into a disjunctive datalog program. To attenuate the increased computational complexity introduced by using disjunctive datalog, we developed a query answering algorithm which supports the principle of "graceful degradation": the user only pays a performance penalty for the features actually used in a knowledge base.

In our future work, we shall try to extend the reduction algorithm to support all of OWL-DL. Furthermore, we are currently implementing the algorithms presented here in KAON2, a new hybrid reasoner, for which we shall conduct a thorough performance evaluation.

Acknowledgements

We thank Stefan Decker for his very valuable comments. This work was partially funded by the EU IST project DIP 507483.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, January 2003.
2. F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int'l Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, Australia, 1991.
3. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

4. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.
5. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.
6. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
7. E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *Proc. of 12th IEEE Symposium on Logic in Computer Science LICS '97*, Warsaw, Poland, 1997.
8. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the Twelfth Int'l World Wide Web Conf. (WWW 2003)*, pages 48–57. ACM, 2003.
9. V. Haarslev and R. Möller. RACER System Description. In *1st Int'l Joint Conf. on Automated Reasoning (IJCAR-01)*, pages 701–706. Springer-Verlag, 2001.
10. I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. 6th Int'l. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, 1998.
11. I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth Int'l World Wide Web Conf.(WWW 2004)*. ACM, 2004.
12. I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In B. Nebel, editor, *Proc. of the 17th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.
13. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
14. U. Hustadt, B. Motik, and U. Sattler. Reasoning for Description Logics around $\mathcal{SHIQ}$ in a Resolution Framework. Technical Report 3-8-04/04, FZI, Karlsruhe, Germany, April 2004. http://www.fzi.de/wim/publikationen.php?id=1172.
15. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proc. of the 9th Conference on Knowledge Representation and Reasoning (KR2004)*. AAAI Press, June 2004.
16. A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.
17. C. Lutz. Description Logics with Concrete Domains—A Survey. In *Advances in Modal Logics*, volume 4. King's College Publications, 2003.
18. B. Motik, A. Maedche, and R. Volz. Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases. In *10th Int'l Workshop on Knowledge Representation meets Databases (KRDB-2003)*, Hamburg, Germany, September 15-16 2003.
19. A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001.
20. P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. OWL Web Ontology Language; Semantics and Abstract Syntax, W3C Candidate Recommendation. http://www.w3.org/TR/owl-semantics/, November 2002.
21. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.
22. M. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. AMS, 1997.