Query Answering for OWL-DL with Rules*

Boris Motik[†] Ulrike Sattler[‡]
Rudi Studer[†]

[†]FZI Research Center for Information Technologies
University of Karlsruhe, Germany
{motik,studer}@fzi.de

[‡]Department of Computer Science,
University of Manchester, UK
sattler@cs.man.ac.uk

June 15, 2005

Abstract

Both OWL-DL and function-free Horn rules are decidable fragments of first-order logic with interesting, yet orthogonal expressive power. A combination of OWL-DL and rules is desirable for the Semantic Web; however, it might easily lead to the undecidability of interesting reasoning problems. Here, we present a *decidable* such combination where rules are required to be *DL-safe*: each variable in the rule is required to occur in a non-DL-atom in the rule body. We discuss the expressive power of such a combination and present an algorithm for query answering in the related logic $\mathcal{SHIQ}$ extended with DL-safe rules, based on a reduction to disjunctive programs.

Keywords

Description logics, rules, decidability, hybrid languages

*Accepted for publishing in Elsevier Web Semantics Journal, Special Issue on Rules for the Semantic Web

1 Introduction

OWL-DL [25] 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, $\mathcal{SHIQ}(\mathbf{D})$ [17, 16], distinguished from $\mathcal{SHOIN}(\mathbf{D})$ mainly by not supporting nominals (concepts containing exactly the specified set of individuals), has been successfully implemented in practical reasoning systems, such as Racer [13] and FaCT [14]. 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 [12]. In contrast, decidable rule-based formalism such as function-free Horn rules[1] do not share this restriction, but lack 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 [15], but this extension is undecidable [15]. 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, so to decide satisfiability (i.e. the existence of a model of a knowledge base), one can restrict the search for a model only to 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 following rule, which obviously has only non-tree models:

$$hasAunt(x, y) \leftarrow hasParent(x, z), hasSibling(z, y), Female(y)$$

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 op-

[1] Throughout this paper, we use "rules" and "clauses" synonymously, following [15].

timizations, 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 [22, 6, 15], 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 such a combined logic is decidable.

Moreover, we present an algorithm for query answering in the extension of $\mathcal{SHIQ}$ with DL-safe rules which is based on a novel technique for reducing $\mathcal{SHIQ}$ knowledge bases to disjunctive programs [20, 18]. 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 initial results are 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 [15] since our approach is fully compatible with the one proposed there. This paper is an extended version of [23]. Due to space constraints, this paper does not contain all technical details and proofs. All technical results have been summarized in [18].

The paper is structured as follows. In Section 2 we give the definitions necessary to understand this paper. In Section 3 we informally discuss the difficulties in combining OWL-DL and rules. In Section 4 we introduce the notion of DL-safe rules and show that query answering for DL-safe rules is decidable. In Section 5 we present a practical algorithm for reasoning in a logic obtained by combining $\mathcal{SHIQ}$ and DL-safe rules. Before we conclude, in Section 6 we give an overview of the related work.

2 Preliminaries

2.1 OWL-DL and its Variants

OWL-DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [15]. Hence, although several syntactic forms 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 existing OWL-DL syntactic forms, see [15].

$\mathcal{SHOIN}(\mathbf{D})$ supports reasoning with concrete datatypes, such as strings or integers. Instead of axiomatizing datatypes in logic, $\mathcal{SHOIN}(\mathbf{D})$ employs a restricted version of the approach from [2], where the properties of concrete datatypes are encapsulated in so-called *concrete domains*, which we introduce below. In the rest, with $\mathbf{x}$ we denote a vector of variables $x_1, \ldots, x_n$ and, for a function δ, with $\delta(\mathbf{x})$ we denote the application of δ to each element x_i of $\mathbf{x}$.

Definition 1. *A concrete domain* $\mathbf{D}$ *is a pair* $(\triangle_{\mathbf{D}}, \Phi_{\mathbf{D}})$*, where* $\triangle_{\mathbf{D}}$ *is a set, called the* domain *of* $\mathbf{D}$*, and* $\Phi_{\mathbf{D}}$ *is a set of* predicate names*. Each* $d \in \Phi_{\mathbf{D}}$ *is associated with an arity* n *and an* extension $d^{\mathbf{D}} \subseteq \triangle_{\mathbf{D}}^n$*. A concrete domain* $\mathbf{D}$ *is* admissible *if (*i*)* $\Phi_{\mathbf{D}}$ *is closed under negation, i.e. for each* $d \in \Phi_{\mathbf{D}}$*, there exists* $\overline{d} \in \Phi_{\mathbf{D}}$ *with* $\overline{d}^{\mathbf{D}} = \triangle_{\mathbf{D}}^n \setminus d^{\mathbf{D}}$*, (*ii*)* $\Phi_{\mathbf{D}}$ *contains a unary predicate* $\top_{\mathbf{D}}$ *interpreted as* $\triangle_{\mathbf{D}}$*, (*iii*)* $\Phi_{\mathbf{D}}$ *contains a binary predicate* $\approx_{\mathbf{D}}$ *interpreted as* $\{(x, y) \mid x = y\}$*, and (*iv*)* $\mathbf{D}$*-satisfiability of finite conjunctions of the form* $\bigwedge_{i=1}^{n} d_i(\mathbf{x_i})$ *is decidable. The latter is the case if an assignment* δ *of variables to elements of* $\triangle_{\mathbf{D}}$ *exists, such that* $\delta(\mathbf{x_i}) \in d_i^{\mathbf{D}}$*, for each* $1 \leq i \leq n$*.*

Notice that, since descriptions logics considered in this paper can enforce equality between concrete terms, we extend the notion of the admissibility from [2] with Condition (*iii*). To simplify extending first-order logic with a concrete domain, we assume the existence of two sorts: c for the concrete domain, and a for all other objects of the so-called *abstract* domain. To distinguish the sorts syntactically, we denote the variables (function symbols) of sort c as x^{c} (f^{c}). We assume that (*i*) the predicates from $\Phi_{\mathbf{D}}$ are contained in a first-order signature Σ, (*ii*) they have only arguments of sort c and (*iii*) for each function symbol f, no argument is of sort c. Under these assumptions, a first-order interpretation I is called a $\mathbf{D}$-*interpretation* if all terms of sort c are interpreted as elements of $\triangle_{\mathbf{D}}$ and the extension of each concrete predicate d is $d^{\mathbf{D}}$. The notions of a model, satisfiability and entailment carry over to the first-order logic with a concrete domain in the obvious way. When ambiguities do not arise, we do not stress $\mathbf{D}$ in "$\mathbf{D}$-model", "$\mathbf{D}$-satisfiability" etc.

We now define the description logic $\mathcal{SHOIQ}(\mathbf{D})$— a generalization of all different description logics which we consider in this paper.

Definition 2. *For* N_{R_a} *a set of abstract role names, the set of* $\mathcal{SHOIQ}(\mathbf{D})$ abstract roles *is the set* $N_{R_a} \cup \{R^- \mid R \in N_{R_a}\}$*. Let* $\mathsf{Inv}(R) = R^-$ *and* $\mathsf{Inv}(R^-) = R$ *for* $R \in N_{R_a}$*. Analogously, let* N_{R_c} *be the set of concrete roles.*[2] *A* $\mathcal{SHOIQ}(\mathbf{D})$ RBox $KB_{\mathcal{R}}$ *over* N_{R_a} *and* N_{R_c} *is a finite set of transitivity axioms* $\mathsf{Trans}(R)$ *and role inclusion axioms* $R \sqsubseteq S$ *and* $T \sqsubseteq U$*, such that* $R \sqsubseteq S \in KB_{\mathcal{R}}$ *implies* $\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \in KB_{\mathcal{R}}$*, and* $\mathsf{Trans}(R) \in KB_{\mathcal{R}}$

[2] We do not distinguish between concrete role names and concrete roles, since inverse concrete roles do not make sense semantically and are therefore not supported.

implies $\mathsf{Trans}(\mathsf{Inv}(R)) \in KB_{\mathcal{R}}$*, for abstract roles* R *and* S*, and concrete roles* T *and* U*.*

Let $\sqsubseteq^*$ *denote the reflexive-transitive closure of* $\sqsubseteq$*. A role* R *is* transitive *if* $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$ *for some* S *with* $S \sqsubseteq^* R$ *and* $R \sqsubseteq^* S$*;* R *is* simple *if there is no role* S *such that* $S \sqsubseteq^* R$ *and* S *is transitive; and* R *is* complex *if it is not simple.*

Let N_C *be a set of* atomic *concept names,* N_{I_a} *a set of* abstract individuals*, and* N_{I_c} *a set of* concrete individuals*. The set of* $\mathcal{SHOIQ}(\mathbf{D})$ concepts *is built by the following syntactic rules, where* A *is a concept name,* $C_{(i)}$ *a* $\mathcal{SHOIQ}(\mathbf{D})$ *concept,* R *an abstract role,* S *a simple abstract role,* $T_{(i)}$ *a concrete role,* n *an integer,* d *a concrete predicate,* $a_{(i)}$ *an abstract individual,* $c^{\mathsf{c}}_{(i)}$ *a concrete individual, and* $\bowtie \in \{\leq, \geq\}$*:*

$$\begin{array}{rcl} C & \rightarrow & \top \mid \bot \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \bowtie n\, S.C \mid \\ & & \{a_1, \ldots, a_n\} \mid \bowtie n\, T \mid \exists T_1, \ldots, T_n.D \mid \forall T_1, \ldots, T_n.D \\ D & \rightarrow & d \mid \{c^{\mathsf{c}}_1, \ldots, c^{\mathsf{c}}_n\} \end{array}$$

A $\mathcal{SHOIQ}(\mathbf{D})$ TBox $KB_{\mathcal{T}}$ *over* N_C *and* $KB_{\mathcal{R}}$ *is a finite set of* concept inclusion axioms $C \sqsubseteq D$*, where* C *and* D *are* $\mathcal{SHOIQ}(\mathbf{D})$ *concepts.*

A $\mathcal{SHOIQ}(\mathbf{D})$ ABox $KB_{\mathcal{A}}$ *is a set of concept and role* membership axioms $C(a)$*,* $R(a, b)$*,* $T(a, b^{\mathsf{c}})$*, and* (in)equality axioms $a^{(\mathsf{c})} \circ b^{(\mathsf{c})}$*, where* $\circ \in \{\approx, \not\approx\}$*,* C *is a* $\mathcal{SHOIQ}(\mathbf{D})$ *concept,* R *an abstract role,* T *a concrete role,* a *and* b *abstract individuals, and* a^{c} *and* b^{c} *concrete individuals.*

A $\mathcal{SHOIQ}(\mathbf{D})$ knowledge base KB *is a triple* $(KB_{\mathcal{R}}, KB_{\mathcal{T}}, KB_{\mathcal{A}})$*, where* $KB_{\mathcal{R}}$ *is an RBox,* $KB_{\mathcal{T}}$ *is a TBox, and* $KB_{\mathcal{A}}$ *is an ABox.*

$\mathcal{SHOIN}(\mathbf{D})$ is obtained from $\mathcal{SHOIQ}(\mathbf{D})$ by allowing only number restrictions of the form $\bowtie n\, R.\top$ (usually written as $\bowtie n\, R$). $\mathcal{SHIQ}(\mathbf{D})$ is obtained from $\mathcal{SHOIQ}(\mathbf{D})$ by not allowing nominal concepts $\{a_1, \ldots, a_n\}$ and datatypes $\{c^{\mathsf{c}}_1, \ldots, c^{\mathsf{c}}_n\}$. $\mathcal{SHOIN}$ and $\mathcal{SHIQ}$ are obtained from $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIQ}(\mathbf{D})$ by disallowing concrete domains. Finally, $\mathcal{ALCHIQ}$ and $\mathcal{ALCHIQ}(\mathbf{D})$ are obtained from $\mathcal{SHIQ}$ and $\mathcal{SHIQ}(\mathbf{D})$ by disallowing transitivity axioms.

Since the algorithms we present in Section 5 are based on resolution, instead of using a direct model-theoretic semantics [17], we present an equivalent semantics by translation into multi-sorted first-order logic.

Definition 3. *The semantics of a* $\mathcal{SHOIQ}(\mathbf{D})$ *knowledge base* KB *is given by the mapping* π *in Table 1 which transforms* KB *into a first-order formula. Each atomic concept is mapped into a unary predicate, each abstract role is mapped into a binary predicate with arguments of sort* $\mathsf{a} \times \mathsf{a}$ *and each concrete role is mapped into a binary predicate with arguments of sort* $\mathsf{a} \times \mathsf{c}$*. The basic inference problem for* $\mathcal{SHOIQ}(\mathbf{D})$ *is checking* KB satisfiability*, i.e. determining whether a first-order model of* $\pi(KB)$ *exists.*

Other interesting inference problems, such as concept satisfiability, concept subsumption and instance checking can be reduced to satisfiability by standard transformations, cf. [1, ch. 2].

2.2 Rules

We use the standard definitions for rules. Let N_P be a set of predicate symbols. 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 $P \in N_P$ is a predicate symbol and s_i are terms. A *disjunctive rule* has the form

$$A_1 \vee \ldots \vee A_m \leftarrow B_1, \ldots, B_n$$

where A_i and B_j are atoms; the set of atoms A_i is called the *rule head*, and the set of all B_j is called the *rule body*. A *non-disjunctive rule* is a rule with $m = 1$. Unless explicitly stated otherwise, the term "rule" refers to a non-disjunctive rule. A *program* P is a finite set of disjunctive rules. For the semantics, we define the above rule to be equivalent to the clause $A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n$. This yields a monotonic function-free formalism compatible with the one from [15].

2.3 Basic Superposition

Our algorithms from Section 5 are based on *basic superposition* — a calculus optimized for theorem proving with equality [4, 24], which we outline in the rest of this section.

We assume the standard notions of first-order clauses with equality: all existential quantifiers have been eliminated using Skolemization; all remaining variables are universally quantified; we only consider the equality predicate, i.e. all non-equational literals A are encoded as $A \approx \top$ in a multi-sorted setting; and we treat $\approx$ as having built-in symmetry. Moreover, we assume the reader to be familiar with standard resolution [3].

Basic superposition is an optimized version of superposition which prohibits superposition into terms introduced by unification in inference steps applied so far. Its inferences rules are formalized by distinguishing two parts of a clause: (i) the *skeleton* clause C and (ii) the *substitution* σ representing the cumulative effects of all unifications. Such a representation of a clause $C\sigma$ is called a *closure*, and is written as $C \cdot \sigma$. A closure can conveniently be represented by *marking* each term in $C\sigma$ occurring at a variable position of C with $[\,]$ if this variable is in the domain of σ. Any position at or beneath a marked position is called a *substitution position*.

The calculus requires two parameters. The first is an *admissible* ordering on terms $\succ$, i.e. a *reduction ordering* total on ground terms. If $\succ$ is total on non-ground terms (as it is the case in this paper), it can be extended to literals by associating, with each literal $L = s \circ t$, $\circ \in \{\approx, \not\approx\}$, a complexity

Table 1: Translation of $\mathcal{SHOIQ}(\mathbf{D})$ into FOL

Mapping Concepts to FOL	
$\pi_y(\top, X)=\top$	$\pi_y(\bot, X)=\bot$
$\pi_y(A, X)=A(X)$	$\pi_y(\neg C, X)=\neg\pi_y(C, X)$
$\pi_y(C \sqcap D, X)=\pi_y(C, X) \wedge \pi_y(D, X)$	$\pi_y(C \sqcup D, X)=\pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X)=\forall y : R(X, y) \rightarrow \pi_x(C, y)$	$\pi_y(\exists R.C, X)=\exists y : R(X, y) \wedge \pi_x(C, y)$
$\pi_y(\{a_1 \ldots, a_n\}, X)=X \approx a_1 \vee \ldots \vee X \approx a_n$	$\pi_y(\{c_1^{\mathsf{c}}, \ldots, c_n^{\mathsf{c}}\}, X)=X \approx_{\mathbf{D}} c_1^{\mathsf{c}} \vee \ldots \vee X \approx_{\mathbf{D}} c_n^{\mathsf{c}}$
$\pi_y(d, X_1, \ldots, X_m)=$	$d(X_1, \ldots, X_m)$
$\pi_y(\leq n\, R.C, X)=$	$\forall y_1, \ldots, y_{n+1} : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \rightarrow \bigvee y_i \approx y_j$
$\pi_y(\geq n\, R.C, X)=$	$\exists y_1, \ldots, y_n : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \wedge \bigwedge y_i \not\approx y_j$
$\pi_y(\forall T_1, \ldots, T_m.D, X)=$	$\forall y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \rightarrow \pi_x(D, y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$
$\pi_y(\exists T_1, \ldots, T_m.D, X)=$	$\exists y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \wedge \pi_x(D, y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$
$\pi_y(\leq n\, T, X)=$	$\forall y_1^{\mathsf{c}}, \ldots, y_{n+1}^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \rightarrow \bigvee y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$
$\pi_y(\geq n\, T, X)=$	$\exists y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \wedge \bigwedge y_i^{\mathsf{c}} \not\approx_{\mathbf{D}} y_j^{\mathsf{c}}$
Mapping Axioms and KB to FOL	
$\pi(C \sqsubseteq D)=$	$\forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$
$\pi(R \sqsubseteq S)=$	$\forall x, y : R(x, y) \rightarrow S(x, y)$
$\pi(\mathsf{Trans}(R))=$	$\forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
$\pi(C(a))=$	$\pi_y(C, a)$
$\pi(R(a, b))=$	$R(a, b)$
$\pi(a^{(\mathsf{c})} \circ b^{(\mathsf{c})})=$	$a^{(\mathsf{c})} \circ_{(\mathbf{D})} b^{(\mathsf{c})}$ for $\circ \in \{\approx, \not\approx\}$
$\pi(KB)=$	$\bigwedge_{R \in N_R} \forall x, y : R(x, y) \leftrightarrow R^-(y, x) \wedge \bigwedge_{\alpha \in KB_{\mathcal{R}} \cup KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} \pi(\alpha)$

X is a meta variable and is substituted with the actual variable. π_x is obtained from π_y by simultaneously substituting all $y_{(i)}$ with $x_{(i)}$ and π_y with π_x, and vice versa.

Table 2: Inference Rules of the $\mathcal{BS}$ Calculus

Positive superposition: $\dfrac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \approx v) \cdot \rho}{(C \vee D \vee w[t]_p \approx v) \cdot \theta}$	(i) $\sigma = \mathsf{MGU}(s\rho, w\rho\vert_p)$ and $\theta = \rho\sigma$, (ii) $t\theta \not\preceq s\theta$ and $v\theta \not\preceq w\theta$, (iii) $(s \approx t) \cdot \theta$ is strictly eligible for superposition, (iv) $(w \approx v) \cdot \theta$ is strictly eligible for superposition, (v) $s\theta \approx t\theta \not\preceq w\theta \approx v\theta$, (vi) $w\vert_p$ is not a variable.
Negative superposition: $\dfrac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \not\approx v) \cdot \rho}{(C \vee D \vee w[t]_p \not\approx v) \cdot \theta}$	(i) $\sigma = \mathsf{MGU}(s\rho, w\rho\vert_p)$ and $\theta = \rho\sigma$, (ii) $t\theta \not\preceq s\theta$ and $v\theta \not\preceq w\theta$, (iii) $(s \approx t) \cdot \theta$ is strictly eligible for superposition, (iv) $(w \not\approx v) \cdot \theta$ is eligible for resolution, (v) $w\vert_p$ is not a variable.
Reflexivity resolution: $\dfrac{(C \vee s \not\approx t) \cdot \rho}{C \cdot \theta}$	(i) $\sigma = \mathsf{MGU}(s\rho, t\rho)$ and $\theta = \rho\sigma$, (ii) $(s \not\approx t) \cdot \theta$ is eligible for resolution.
Equality factoring: $\dfrac{(C \vee s \approx t \vee s' \approx t') \cdot \rho}{(C \vee t \not\approx t' \vee s' \approx t') \cdot \theta}$	(i) $\sigma = \mathsf{MGU}(s\rho, s'\rho)$ and $\theta = \rho\sigma$, (ii) $t\theta \not\preceq s\theta$ and $t'\theta \not\preceq s'\theta$, (iii) $(s \approx t) \cdot \theta$ is eligible for superposition.
Ordered hyperresolution: $\dfrac{E_1 \ldots E_n \quad E}{(C_1 \vee \ldots \vee C_n \vee D) \cdot \theta}$	(i) E_i are of the form $(C_i \vee A_i) \cdot \rho$, for $1 \leq i \leq n$, (ii) E is of the form $(D \vee \neg B_1 \vee \ldots \vee \neg B_n) \cdot \rho$, (iii) σ is the most general substitution such that $A_i\theta = B_i\theta$ for $1 \leq i \leq n$, and $\theta = \rho\sigma$, (iv) $A_i \cdot \theta$ is strictly eligible for superposition, (v) $\neg B_i \cdot \theta$ are selected, or nothing is selected, $i = 1$ and $\neg B_1 \cdot \theta$ is maximal w.r.t. $D \cdot \theta$.

measure $c_L = (\max(s,t), p_L, \min(s,t))$, where p_L is 1 if $\circ$ is $\approx$, and 0 otherwise. Now $L_1 \succ L_2$ iff $c_{L_1} \succ c_{L_2}$, where c_{L_i} are compared lexicographically, with $1 \succ 0$. The second parameter of the calculus is a *selection function* which selects an arbitrary set of negative literals.

The basic superposition calculus is a refutation procedure: if a set of closures N is *saturated up to redundancy*, then it is unsatisfiable if and only if it contains the empty closure. The set of closures N is saturated up to redundancy if all inferences from premises in N are redundant in N. A literal $L \cdot \sigma$ is (strictly) maximal w.r.t. a closure $C \cdot \sigma$ if no $L' \in C$ exists, such that $L'\sigma \succ L\sigma$ ($L'\sigma \succeq L\sigma$). A literal $L \cdot \sigma$ is *(strictly) eligible for superposition* in $(C \vee L) \cdot \sigma$ if there are no selected literals in $(C \vee L) \cdot \sigma$ and $L \cdot \sigma$ is (strictly) maximal w.r.t. $C \cdot \sigma$; $L \cdot \sigma$ is *eligible for resolution* in $(C \vee L) \cdot \sigma$ if it is selected in $(C \vee L) \cdot \sigma$ or there are no selected literals in $(C \vee L) \cdot \sigma$ and $L \cdot \sigma$ is maximal w.r.t. $C \cdot \sigma$. We denote basic superposition with $\mathcal{BS}$ and present its inference rules in Table 2. The ordered hyperresolution rule is a "macro" inference, combining negative superposition and reflexivity resolution; E is called the *main premise*, and E_i are called the *side premises*.

3 Reasons for the Undecidability of OWL-DL with Rules

In [15], an extension of OWL-DL with rules was presented, by requiring that $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. In other words, integration of OWL-DL and rules is achieved by simply allowing concepts and roles to be used in rules as unary and binary atoms, respectively. Furthermore, it was shown that such an extension leads to undecidability of the following problem: 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 are also undecidable. Investigating this proof and the ones in [22] 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 3. 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, who 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, who is a person.

Let us now check whether $KB \models \mathit{Grandchild}(\mathit{Jane})$; this is the case if and only if $KB \cup \{\neg \mathit{Grandchild}(\mathit{Jane})\}$ 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

Table 3: Example Knowledge Base

$Person(Peter)$	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.

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 completeness 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 [29]. 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 non-tree models). Even if such a tree model is infinite, we can *wind* this infinite tree model into a finite structure. In our example, since KB does not require each father in the chain to be distinct (i.e. there is no axiom requiring the role *father* to be acyclic), the model in Figure 1, lower part (b) is the result of "winding" an infinite tree into a "nice", finite model. Due to their regular structure, these "windings" of tree models can be easily constructed in an automated way. To understand why every satisfiable $\mathcal{SHIQ}$ knowledge base has a tree model [17], consider the mapping π in Table 1 more closely (we ignore 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:

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

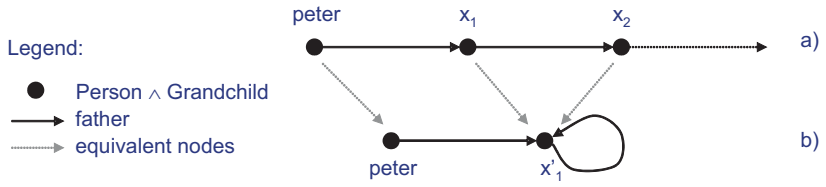

Figure 1: Two Similar Models

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 which are explicitly introduced and are given a name in the knowledge base. Reasoning can be performed by *grounding* the rules, i.e. replacing the variables in the rules with all individuals from the knowledge base in all possible ways. Through grounding, first-order reasoning becomes propositional, since a ground rule is essentially equivalent to a propositional clause. For a finite program, the number of ground rules is also finite, and satisfiability of a set of propositional clauses is decidable. Hence, the rules, such as the one defining $hasAunt(x, y)$ from the introduction, are allowed to enforce arbitrary but finite, non-tree models, and not only "nice" models.

Now let us see what happens if we extend a description logic such as $\mathcal{SHIQ}$ 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 [22, 15].

4 DL-safe Rules

As a reaction to the observations in Section 3, in this section we first formalize the interface between description logics and rules, and then, to achieve decidability, we define the notion of *DL-safe* rules and discuss its benefits and drawbacks. Finally, we show that query answering in $\mathcal{SHOIN}$ with DL-safe rules is decidable.

4.1 Combining Description Logics and Rules

Definition 4 (DL Rules)**.** *Let KB be a $\mathcal{SHOIQ}(\mathbf{D})$ knowledge base and let N_P be the set of predicate symbols such that $\{\approx\} \cup N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. For s and t constants or variables, 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* non-DL-atom *is an atom with a predicate from $N_P \setminus (N_C \cup N_{R_a} \cup N_{R_c} \cup \{\approx\})$. A (disjunctive)* DL rule *is a (disjunctive) rule over N_P. A DL program is a set of (disjunctive) DL rules.*

The semantics of the combined knowledge base *(KB, P), where KB is a $\mathcal{SHOIQ}(\mathbf{D})$ knowledge base and P is a DL program, is given by translation into first-order logic as $\pi(KB) \cup P$, where each rule $A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$ is treated as a clause $A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m$. The main inferences*

in (KB, P) *are* satisfiability checking*, i.e. determining whether a first-order model of* $\pi(KB) \cup P$ *exists, and* query answering*, i.e. determining whether* $\pi(KB) \cup P \models \alpha$ *for a ground atom* α*, written as* $(KB, P) \models \alpha$*.*

A few remarks regarding Definition 4 are in order.

Relationship with Existing Formalisms. The above definition yields a formalism compatible with the ones from [15, 22]. The main difference from [15] is that we allow non-DL-atoms to occur in a rule, and that we only allow atomic concepts to occur in a rule. The latter is a technical assumption and is not really a restriction: for a complex concept C, one can always introduce a new atomic concept A_C, add the axioms $A_C \sqsubseteq C$ and $C \sqsubseteq A_C$ to the TBox, and use A_C in the rule. This transformation is obviously linear in the size of P.

Decidability. Since the formalism is compatible with [15], we immediately have that the reasoning with combined knowledge bases is undecidable. To achieve decidability, we introduce the notion of DL-safety in Subsection 4.2.

Minimal vs. First-order Models. Rules are usually interpreted under *minimal model* semantics, i.e. only models minimal w.r.t. set inclusion are considered. We write $P \models_c \alpha$ if a formula α is true in all minimal models of P. However, in Definition 5 we assume the standard first-order semantics for rules, where $P \models \alpha$ means that α is true in all models of P. We briefly discuss the differences between these two approaches, and their practical consequences.

Assume that α is a positive ground atom. It is easy to see that in such a case, $P \models \alpha$ if and only if $P \models_c \alpha$. Namely, if α is true in each model of P, it is true in each minimal model of P as well, and vice versa. Therefore, for entailment of positive ground atoms, it is not important whether the semantics of P is defined w.r.t. minimal or w.r.t. general first-order models.

Assume now that α is a negative ground atom. In this case, there is a difference between minimal model semantics and first-order semantics, as shown by the following example. For $\alpha = \neg A(b)$ and $P = \{A(a)\}$, it is clear that $P \not\models \alpha$. Namely, $\neg A(b)$ is not explicitly derivable from the facts in P: $M_1 = \{A(a), A(b)\}$ is a first-order model of P and α is false in M_1. However, P has exactly one minimal model $M_2 = \{A(a)\}$ and $\neg A(b)$ is obviously true in M_2, so $P \models_c \alpha$.

The choice of semantics also affects concept subsumption: assume that $\alpha = \forall x : C(x) \rightarrow D(x)$, and $P = \{C(a), D(a)\}$. Similarly as above, $P \not\models \alpha$: just consider a model $M_1 = \{C(a), D(a), C(b)\}$ of P in which α is false. However, the only minimal model of P is $M_2 = \{C(a), D(a)\}$ and α is true in M_2, so $P \models_c \alpha$. The distinction between minimal models and general first-order models fundamentally changes the computational properties

of concept subsumption: equivalence of general programs under minimal model semantics is undecidable [27] whereas, under first-order semantics, it is decidable and can be reduced to satisfiability checking using standard transformations.

To summarize, the difference between first-order and minimal model semantics is not relevant for query answering if queries are positive atoms; however, it is relevant for queries which involve negation or for concept subsumption. Negative queries are usually considered in a more general framework of *negation-as-failure*, where negation is interpreted as failure to prove a query, thus yielding a *non-monotonic* formalism. Whereas non-monotonic features are certainly very important for the Semantic Web, we do not address them in this work. Instead, our results are an initial step towards providing a practical hybrid knowledge representation formalism integrating description logics and rules. We also believe that our work may be used as a basis for future non-monotonic extensions.

4.2 DL-safety Restriction

We now introduce DL-safety restriction as one possible way to make reasoning with DL rules decidable.

Definition 5 (DL-safe Rules)**.** *A (disjunctive) DL rule r is* DL-safe *if each variable occurring in r also occurs in a non-DL-atom in the body of r. A (disjunctive) program P is DL-safe if all its rules are DL-safe.*

DL-safety is similar to 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 ensures 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-atoms $\mathcal{O}(x)$ and $\mathcal{O}(y)$ to the body of the rule, and by adding a fact $\mathcal{O}(a)$ for each individual a occurring in KB and P. Thus, the above rule becomes

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y), \mathcal{O}(x), \mathcal{O}(y)$$

This rule is obviously DL-safe. In Subsection 4.3 we discuss the consequences that this transformation has on the semantics.

Table 4: Example with DL-safe Rules

$Person \sqsubseteq \exists father.Person$	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.
$father \sqsubseteq parent$	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 individual α in the ABox	Enumeration of all ABox individuals.

4.3 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, consider the axioms from Table 4. 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 variables x, y and z do 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*. Now *Cain* and *Abel* are children of *Adam*, and *Cain hates Abel*, so *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.

Consider now the DL-safe rule defining *BadChild*′: 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, 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 intuitive 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 thus to conclude 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 last *KB* part about *Oedipus*: he is a *GoodChild* or a *BadChild′*, but we do not know exactly which is true. Either way, one of the rules derives that *Oedipus* is a *Child*, so we have $(KB, P) \models Child(Oedipus)$. This cannot be derived by applying the rules defining *Child* to the consequences of *KB* since $KB \not\models GoodChild(Oedipus)$ and $KB \not\models BadChild'(Oedipus)$.

The impact of the DL-safety restriction in practice depends a lot on the type of the application. For applications relying mainly on extensional reasoning (such as e.g. metadata-based information retrieval), we believe that DL-safety is not a serious restriction. In such applications, the universe of discourse is usually limited to the known objects, so DL-safe rules can draw all or most relevant conclusions. On the contrary, in applications requiring intensional reasoning (such as e.g. natural language processing), DL-safety is a much more severe restriction, as many conclusions drawn involve unnamed objects.

4.4 Decidability of Query Answering

We now show that checking satisfiability of a $\mathcal{SHOIN}$ knowledge base extended with DL-safe rules is decidable. The proof is by a non-deterministic reduction to checking satisfiability of a $\mathcal{SHOIN}$ knowledge base without rules.

Theorem 1. *For any $\mathcal{SHOIN}$ knowledge base KB and any DL-safe program P, checking if (KB, P) is satisfiable is decidable.*

Proof. 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 if and only if $\pi(KB) \cup P^g$ 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 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 is 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 $\pi(KB) \cup L^c$. Now $\pi(KB) \cup P^g$ is satisfiable if and only if 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 subset of ground literals of L^c involving DL predicates. Then $\pi(KB) \cup L^c$ is unsatisfiable if and only if 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 syntactically, 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 $\neg R(a, b)$, which can be transformed into equivalent $\mathcal{SHOIN}$ ABox assertions $(\forall R.\neg\{b\})(a)$. Thus we have reduced query answering to deciding satisfiability of a $\mathcal{SHOIN}$ knowledge base.

The latter 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 [18] 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 [10]. □

Since the semantic of (KB, P) is compatible with standard first-order semantics, we have that $(KB, P) \models \alpha$ if and only if $(KB, P \cup \{\leftarrow \alpha\})$ is unsatisfiable, for any ground atom α. Hence, Theorem 1 shows decidability of query answering as well.

The same non-deterministic reduction of ground DL-safe rules to sets of ground literals is applicable even if KB is a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base. Hence, we strongly believe that Theorem 1 also holds for $\mathcal{SHOIN}(\mathbf{D})$, since the proof that $\mathcal{SHOIN}$ is decidable should be easily adaptable to $\mathcal{SHOIN}(\mathbf{D})$.

5 Query Answering with DL-safe Rules

The proof of Theorem 1 gives a procedure for query answering in the full combination of OWL-DL and DL-safe rules. However, this procedure is likely to be hopelessly inefficient in practice, due to the huge amount of don't-known non-determinism. At least to a certain extent, such rather blind guessing seems unavoidable. Namely, in [28] it was shown that the combination of nominals, inverse roles, and number restrictions yields an increase in complexity from ExpTime to NExpTime. Therefore, in this section we describe a *practical* reasoning algorithm for the following fragment: (*i*) the description logic is $\mathcal{SHIQ}$, 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 disjunctive program which entails the same set of ground facts as the original knowledge base. For unary coding of numbers, this algorithm runs in deterministic exponential time, which makes it optimal since $\mathcal{SHIQ}$ is ExpTime-complete [28]. Furthermore, DL-safe rules (with the above restriction to concepts and simple roles) can simply be appended to the program obtained by the reduction.

The full presentation of the algorithm and a proof of its correctness are technically involved and lengthy [18, 20]. Here, we just provide an overview of the procedure, without going into details.

The algorithm in [18] is capable of handling $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. The presence of datatypes does not significantly affect the algorithm presented in this paper: the only change is to the basic superposition calculus, which is extended with a so-called *concrete domain resolution rule* from [19]. This rule combines concrete domain reasoning with logical reasoning to provide a **D**-refutation procedure. Since its presentation is rather technical and not essential for the ideas described here, for details please refer to [18, 19].

5.1 Overview

For a $\mathcal{SHIQ}$ knowledge base KB, our goal is to obtain a disjunctive program $\mathsf{DD}(KB)$, such that $KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, for α of the form $R(a, b)$ or $A(a)$. Thus, $\mathsf{DD}(KB)$ can be used for answering queries in KB.

The intuition behind our reduction is the following: let us assume that unsatisfiability of KB can be decided using some sound and complete calculus $\mathcal{C}$. Our goal is to simulate each inference step of $\mathcal{C}$ on KB by a sound inference step on $\mathsf{DD}(KB)$. Hence, each proof by $\mathcal{C}$ in KB can be reduced to a proof in $\mathsf{DD}(KB)$. Furthermore, the program $\mathsf{DD}(KB)$ should allow simulating the inferences in the other direction as well, i.e. it should be possible to reduce each proof in $\mathsf{DD}(KB)$ by a sound and complete calculus $\mathcal{C}'$ to a proof in KB. Then, KB is satisfiable if and only if $\mathsf{DD}(KB)$ is satisfiable.

To operationalize this idea, we use an instance of basic superposition for $\mathcal{C}$ and, in Section 5.2, we show an algorithm for deciding satisfiability of KB

by $\mathcal{BS}^{+}_{DL}$, a parametrization of $\mathcal{BS}$. Next, in Section 5.3 we show how this decision procedure can be used to obtain the desired reduction. In Section 5.4 we show that DL-safe rules can simply be appended to $\mathsf{DD}(KB)$. In Section 5.5 we show how to compute non-ground query answers. Finally, in Section 5.6 we present an example knowledge base along with the steps performed by the reduction algorithm.

5.2 Deciding $\mathcal{SHIQ}$ by Basic Superposition

Satisfiability of a $\mathcal{SHIQ}$ knowledge base KB can be decided in the framework of resolution as follows.

Eliminating Transitivity Axioms. A minor problem in deciding satisfiability of KB are the transitivity axioms, which, in their clausal form, do not contain so-called *covering* literals (i.e. literals containing all variables of a clause). Such clauses are known to be difficult to handle, so we preprocess KB into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$. In short, this transformation replaces each transitivity axiom $\mathsf{Trans}(S)$ with axioms of the form $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each R with $S \sqsubseteq^{*} R$ and C a concept occurring in KB. This transformation is polynomial. Whereas KB and $\Omega(KB)$ entail the same sets of ground facts concerning simple roles, they do not entail the same sets of ground facts concerning complex roles. This is the reason for allowing only simple roles to occur in DL-safe rules.

Preprocessing. We translate $\Omega(KB)$ into a first-order formula $\pi(KB)$ according to Table 1. Assuming unary coding of numbers, $\pi(KB)$ can be computed in polynomial time. To transform $\pi(KB)$ into a set of closures $\Xi(KB)$, we apply the well-known *structural transformation* [26]. Roughly speaking, the structural transformation introduces a new name for each non-atomic subconcept of KB. For example, in the axiom $C \sqsubseteq \exists R.(\forall S.C)$, a new concept Q is introduced for the subconcept $\forall S.C$, and the above axiom is replaced with axioms $C \sqsubseteq \exists R.Q$ and $Q \sqsubseteq \forall S.C$. It is well-known that $\pi(KB)$ and $\Xi(KB)$ are equisatisfiable, and that $\Xi(KB)$ can be computed in polynomial time [26].

For any KB, the syntactic structure of closures in $\Xi(KB)$ follows the types given in Table 5; we call them $\mathcal{ALCHIQ}$-closures. We use the following notation: for a term t, we denote with $\mathbf{P}(t)$ a disjunction of the form $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$, and we denote with $\mathbf{P}(\mathbf{f}(x))$ a disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$ (notice that this allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals). We use $\langle t \rangle$ to denote that the term t may, but need not be marked. In all closure types, some of the disjuncts may be empty.

Table 5: Types of $\mathcal{ALCHIQ}$-closures

1	$\neg R(x,y) \vee \mathsf{Inv}(R)(y,x)$
2	$\neg R(x,y) \vee S(x,y)$
3	$\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$
4	$\mathbf{P^f}(x) \vee R([f(x)], x)$
5	$\mathbf{P_1}(x) \vee \mathbf{P_2}(\langle \mathbf{f}(x) \rangle) \vee \bigvee \langle f_i(x) \rangle \approx / \not\approx \langle f_j(x) \rangle$
6	$\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(\langle \mathbf{f}([g(x)]) \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$ where t_i and t_j are either of the form $f([g(x)])$ or of the form x
7	$\mathbf{P_1}(x) \vee \bigvee \neg R(x, y_i) \vee \mathbf{P_2}(\mathbf{y}) \vee \bigvee y_i \approx y_j$
8	$\mathbf{R}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle) \vee \mathbf{P}(\langle \mathbf{t} \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$ where t, t_i and t_j are either some constant b or a functional term $f_i([a])$
Conditions:	
(i):	In any term $f(t)$, the inner term t occurs marked.
(ii):	In all positive equality literals with at least one function symbol, both sides are marked.

Decomposition. As discussed in [21, 18], if KB contains number restrictions on roles that have subroles, saturating $\Xi(KB)$ by $\mathcal{BS}$ need not terminate. To remedy that, we introduce *decomposition* — an additional inference rule which transforms some conclusions of $\mathcal{BS}$ in a way that guarantees termination. More precisely, any conclusion derived by $\mathcal{BS}$ of the form as specified below left is replaced with the two closures on its right, where t is an arbitrary term, and the predicate $Q_{S,f}$ is a predicate not occurring in $\Xi(KB)$ and is unique for a pair of role and function symbols S and f:

$$D \cdot \rho \vee R([t], [f(t)]) \quad \rightsquigarrow \quad \begin{array}{rcl} D \cdot \rho & \vee & Q_{R,f}([t]) \\ \neg Q_{R,f}(x) & \vee & R(x, [f(x)]) \end{array}$$

$$D \cdot \rho \vee R([f(t)], [t]) \quad \rightsquigarrow \quad \begin{array}{rcl} D \cdot \rho & \vee & Q_{\mathsf{Inv}(R),f}([t]) \\ \neg Q_{\mathsf{Inv}(R),f}(x) & \vee & R([f(x)], x) \end{array}$$

With $\mathcal{BS}^+$ we denote the $\mathcal{BS}$ calculus where decomposition is eagerly applied to the conclusions of all $\mathcal{BS}$ inference. In [21, 18] we have shown that $\mathcal{BS}^+$ is sound and complete, i.e. that a set of closures saturated under $\mathcal{BS}^+$ up to redundancy is unsatisfiable if and only if it contains the empty closure.

Parameters for $\mathcal{BS}^+$. In the rest, $\mathcal{BS}^+_{DL}$ denotes the following parametrization of $\mathcal{BS}^+$. We use a standard *lexicographic path ordering* [24] (LPO) for comparing terms. LPOs are based on a precedence $>_P$ over function, constant, and predicate symbols. If the precedence is total, then the induced LPO is total on ground terms, and is admissible for basic superposition. To decide $\mathcal{ALCHIQ}$, we can use any precedence such that

$f >_P c >_P p >_P Q_{S,f} >_P \top$, for any function symbol f, constant c, predicate symbol p and predicate $Q_{S,f}$. We use the selection function which selects all negative binary literals in a closure.

Termination of $\mathcal{BS}^+_{DL}$ on $\mathcal{ALCHIQ}$ Closures. The following lemma is central to our work. It states that any $\mathcal{BS}^+_{DL}$ inference, when applied to $\mathcal{ALCHIQ}$-closures, produces an $\mathcal{ALCHIQ}$-closure. The proof is by considering all $\mathcal{BS}^+_{DL}$ inferences on all types of $\mathcal{ALCHIQ}$-closures.

Lemma 1. *Let $N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}^+_{DL}$-derivation, where $N_0 = \Xi(KB)$ and C is the conclusion derived from premises in N_i. Then C is either an $\mathcal{ALCHIQ}$-closure or it is redundant in N_i.*

Lemma 1 is crucial to show that saturation of $\Xi(KB)$ by $\mathcal{BS}^+_{DL}$ terminates. Namely, for a finite knowledge base, the number of $\mathcal{ALCHIQ}$-closures is finite: (*i*) the number of variables in closures of types 1 and 2 is two, (*ii*) the number of variables in a closure of type 7 is limited by the maximal number occurring in a number restriction, (*iii*) the depth of a functional term is at most two, so (*iv*) using a finite number of concept and function symbols and a finite number of variables, the number of closures that can be built without repeated literals is finite. In fact, assuming that $|KB|$ represents the size of the knowledge base where numbers are coded in unary, the number of $\mathcal{ALCHIQ}$-closures is exponential in $|KB|$.

Therefore, saturation of $\Xi(KB)$ by $\mathcal{BS}^+_{DL}$ takes at most exponentially many derivation steps in $|KB|$. Namely, by Lemma 1, each $\mathcal{BS}^+_{DL}$ inference produces an $\mathcal{ALCHIQ}$-closure. Hence, after at most an exponential number of steps, all possible $\mathcal{ALCHIQ}$-closures will have been derived, after which any $\mathcal{BS}^+_{DL}$ inference will produce an already derived closure. Since $\mathcal{BS}^+_{DL}$ is a sound and complete calculus, the saturated set contains the empty closure if and only if $\Xi(KB)$ is unsatisfiable. Hence, we have the following result:

Theorem 2. *For an $\mathcal{ALCHIQ}$ knowledge base KB, saturating $\Xi(KB)$ by $\mathcal{BS}^+_{DL}$ with eager application of redundancy elimination rules decides satisfiability of KB and runs in time exponential in $|KB|$ for unary coding of numbers.*

5.3 Reducing KB to a Disjunctive Program

We now show how $\mathcal{BS}^+_{DL}$ can be used to reduce an $\mathcal{ALCHIQ}$ knowledge base KB to a disjunctive program $\mathsf{DD}(KB)$. Using the transformation for eliminating transitivity axioms, this algorithm is easily generalized to the case where KB is a $\mathcal{SHIQ}$ knowledge base.

Saturation of TBox and RBox by $\mathcal{BS}^+_{DL}$. Let $\mathsf{gen}(KB)$ be the set of all closures of the form $\neg Q_{R,f}(x) \vee R(x, [f(x)])$, for each role R and a

function symbol f occurring in $\Xi(KB)$. Intuitively, $\mathsf{gen}(KB)$ is the set of closures that can be introduced by the decomposition rule in the saturation of $\Xi(KB)$. Let $\Gamma_{\mathcal{TR}g} = \Xi(KB_{\mathcal{T}} \cup KB_{\mathcal{R}}) \cup \mathsf{gen}(KB)$. Since the predicates $Q_{R,f}$ do not occur in $\Xi(KB)$, $\Gamma_{\mathcal{TR}g} \cup \Xi(KB_{\mathcal{A}})$ is satisfiable if and only if $\Xi(KB)$ is satisfiable.

The first step in the reduction of KB to a disjunctive program is to saturate $\Gamma_{\mathcal{TR}g}$ — the RBox and TBox closures of $\Xi(KB)$ — by $\mathcal{BS}^+_{DL}$; let Γ' be the set of saturated closures. In this key step of the reduction, we compute all non-ground consequences of KB. As discussed in Section 5.2, Γ' can be computed in time exponential in $|KB|$, contains closures of length polynomial in $|KB|$, and contains at most exponentially many $\mathcal{ALCHIQ}$-closures.

Now $\Xi(KB)$ is satisfiable if and only if $\Gamma' \cup \Xi(KB_{\mathcal{A}})$ is satisfiable, and the latter can be decided by saturating $\Gamma' \cup \Xi(KB_{\mathcal{A}})$ by $\mathcal{BS}^+_{DL}$. We also observe that, since Γ' already contains all non-ground consequences of $\Xi(KB)$ and due to the syntactic form of $\mathcal{ALCHIQ}$-closures, only ground closures of type 8 are derived in the saturation of $\Gamma' \cup \Xi(KB_{\mathcal{A}})$. Furthermore, closures of types 4 and 6 cannot participate in any $\mathcal{BS}^+_{DL}$ inference with a ground closure: ground closures do not contain terms of depth two which would unify with the maximal literal from a closure of type 6, and role literals do not contain functional terms which would unify with a term $f(x)$ from a closure of type 4. Hence, closures of types 4 and 6 can safely be removed from Γ'; with Γ we denote the resulting set of closures.

Elimination of Function Symbols. We have seen that satisfiability of KB can be decided by saturating $\Gamma \cup \Xi(KB_{\mathcal{A}})$ under $\mathcal{BS}^+_{DL}$, where all derived closures are of type 8. Closures in Γ and the closures of type 8 obtained by the saturation can contain functional terms of depth one. To obtain a disjunctive program from Γ, i.e., to remove all functional terms, we perform the transformation described below which allows "simulating" each ground functional term $f(a)$ with a new constant a_f.

More precisely, we define an operator λ on the set of terms, producing a term, as follows, where a_f is a new, globally unique constant (i.e. for a pair of a and f, there is a unique constant a_f), and x_f is a new, globally unique variable:

$$\lambda(t) = \begin{cases} a & \text{if } t = a \\ a_f & \text{if } t = f(a) \\ x & \text{if } t = x \\ x_f & \text{if } t = f(x) \end{cases}$$

We extend λ to $\mathcal{ALCHIQ}$-closures such that, for a closure C, $\lambda(C)$ is the following function-free closure:

1. Each term t in C is replaced with $\lambda(t)$.

2. For each variable x_f introduced in the previous step, the disjunct $\vee\neg S_f(x, x_f)$ is appended to C.

3. If, after steps 1 and 2, some variable x occurs in a positive literal, but not in a negative literal, the disjunct $\vee\neg HU(x)$ is appended to C.

Finally, let $\mathsf{FF}(KB) = \mathsf{FF}_\lambda(KB) \cup \mathsf{FF}_{Succ}(KB) \cup \mathsf{FF}_{HU}(KB) \cup \Xi(KB_\mathcal{A})$ denote the function-free version of $\Xi(KB)$, where FF_λ, FF_{Succ} and FF_{HU} are defined as follows:

$$\begin{aligned}
\mathsf{FF}_\lambda(KB) &= \{\lambda(C) \mid C \in \mathsf{Sat}_\mathsf{R}(\Gamma_{\mathcal{TR}g})\} \\
\mathsf{FF}_{Succ}(KB) &= \{S_f(a, a_f) \mid \text{for each } a \text{ and } f \text{ from } \Xi(KB)\} \\
\mathsf{FF}_{HU}(KB) &= \{HU(a) \mid \text{for each } a \text{ from } \Xi(KB)\} \cup \\
&\quad \{HU(a_f) \mid \text{for each } a \text{ and } f \text{ from } \Xi(KB)\}
\end{aligned}$$

Intuitively, λ replaces each term of the form $f(a)$ with a new constant a_f and, for each constant a and each function symbol f, $\mathsf{FF}(KB)$ contains an assertion $S_f(a, a_f)$ stating that a_f is an f-successor of a. Furthermore, each term $f(x)$ in a closure is replaced with x_f, and the literal $\neg S_f(x, x_f)$ is appended to the closure. Hence, the fact that $f(x)$ is the f-successor of x is encoded in S_f. These transformations are sufficient to simulate ground functional terms with new constants. We explain this on a simple example. Consider the following knowledge base KB:

$$\neg C(x) \vee R(x, f(x)) \tag{1}$$

$$R(a, b) \tag{2}$$

$$\neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 \approx y_2 \tag{3}$$

Furthermore, consider the following $\mathcal{BS}^+_{DL}$ inference in KB (we disregard markers in the conclusion):

$$\neg C(a) \vee f(a) \approx b \qquad \text{resolve } (3)+(1)+(2) \tag{4}$$

The set of function-free closures $\mathsf{FF}(KB)$, computed according to the above definition, consists of the following closures:

$$S_f(a, a_f) \tag{5}$$

$$\neg C(x) \vee R(x, x_f) \vee \neg S_f(x, x_f) \qquad = \lambda(1) \tag{6}$$

$$R(a, b) \qquad = \lambda(2) \tag{7}$$

$$\neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 \approx y_2 \qquad = \lambda(3) \tag{8}$$

Now the above inference of (4) can be simulated in $\mathsf{FF}(KB)$ as shown below, where the inference deriving (9) performs a "lookup" for the successor in the predicate S_f. Observe that (10) corresponds to the result of applying λ

to (4), i.e. it contains a_f instead of $f(a)$. Hence, we say that the functional term $f(a)$ is simulated with a new constant a_f.

(9)	$\neg C(a) \vee R(a, a_f)$	resolve (5)+(6)
(10)	$\neg C(a) \vee a_f \approx b$	resolve (8)+(7)+(9)

By considering all inferences of $\mathcal{BS}^+_{DL}$, it is possible to see that, for each closure C derived in the saturation of $\Gamma \cup \Xi(KB_{\mathcal{A}})$, one may derive a closure $\lambda(C)$ from $\mathsf{FF}(KB)$. Hence, if the empty closure is derived by saturation of $\Gamma \cup \Xi(KB_{\mathcal{A}})$, it can also be derived by saturation of $\mathsf{FF}(KB)$.

In a similar way it is possible to show the converse as well, i.e. if a closure C is derivable by saturating $\mathsf{FF}(KB)$, it is possible to derive the closure D from $\Gamma \cup \Xi(KB_{\mathcal{A}})$ such that $C = \lambda(D)$.

Lemma 2. *KB is unsatisfiable if and only if $\mathsf{FF}(KB)$ is unsatisfiable.*

Conversion to a Disjunctive Program. Since $\mathsf{FF}(KB)$ does not contain functional terms and all its closures are safe, we can rewrite each closure into a disjunctive rule by moving positive literals into the head, and negative literals into the body of the rule. We use $\mathsf{DD}(KB)$ for the result of this rewriting. The following theorem summarizes the properties of $\mathsf{DD}(KB)$:

Theorem 3. *For KB a $\mathcal{SHIQ}$ knowledge base, 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 $S(a, b)$ with 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 $|KB|$.*

5.4 Adding DL-safe Rules

Satisfiability of (KB, P) can be decided by saturating $\Xi(KB) \cup P$ by $\mathcal{BS}^+_{DL}$, where we extend the selection function of $\mathcal{BS}^+_{DL}$ to select all non-DL-atoms. We have the following lemma:

Lemma 3. *For an $\mathcal{ALCHIQ}$ knowledge base KB, saturation of $\Xi(KB) \cup P$ by $\mathcal{BS}^+_{DL}$ decides satisfiability of (KB, P).*

Since each rule $r \in P$ is DL-safe, each variable in r occurs in a negative non-DL-atom, which is selected. Hence, each non-ground rule r can be resolved only on non-DL-atoms. Furthermore, a rule can participate as a side premise in hyperresolution only if it does not contain negative atoms; since rules are safe, side premises are ground. All non-DL-atoms contain initially only constants (i.e. they do not contain functional terms), and all variables from r occur in non-DL-atoms in the body. Hence, a hyperresolution with r produces a ground closure containing only constants. Such a closure can participate in inferences with non-ground closures from $\Xi(KB)$ in exactly the same way as in Lemma 1. Hence, adding DL-safe rules does not change non-ground inferences of $\mathcal{BS}^+_{DL}$, and it simply produces new ground closures.

It is now easy to see that Lemma 2 holds even if DL-safe rules are added since such rules do not participate in non-ground inferences used to compute Γ. Furthermore, for each ground closure C derivable by saturating $\Gamma \cup \Xi(KB_{\mathcal{A}}) \cup P$, the closure $\lambda(C)$ can be produced from $\mathsf{FF}(KB) \cup P$, and vice versa. This yields the following theorem:

Theorem 4. *Let KB be a $\mathcal{SHIQ}$ knowledge base and P a DL-safe disjunctive program. Then (*i*) (KB, P) is satisfiable if and only if $\mathsf{DD}(KB) \cup P$ is satisfiable; and (*ii*) $(KB, P) \models \alpha$ if and only if $\mathsf{DD}(KB) \cup P \models_c \alpha$, for α a DL-atom $A(a)$ or $S(a, b)$ with S a simple role, or α a ground non-DL-atom.*

5.5 Evaluating Queries in a Disjunctive Program

Answering queries in disjunctive programs is computationally more expensive than in non-disjunctive programs [7]. 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 a disjunctive program P, which we outline next. This algorithm allows to compute all answers to a query in one "pass", and does not require testing each possible ground answer separately.

With $P_{\approx}$ we denote the datalog program containing rules (11) – (14), where (11) is instantiated for each individual in P, and (14) is instantiated for each predicate symbol R occurring in P. (Notice that it is not necessary to instantiate (14) for $R = \approx$, since such a rule logically follows from symmetry and transitivity.)

$$a \approx a \tag{11}$$

$$x \approx y \leftarrow y \approx x \tag{12}$$

$$x \approx z \leftarrow x \approx y, y \approx z \tag{13}$$

$$R(x_1, \ldots, x'_i, \ldots, x_n) \leftarrow R(x_1, \ldots, x_i, \ldots, x_n), x_i \approx x'_i \tag{14}$$

It is well-known [9] that $P \cup P_{\approx}$, where $\approx$ is treated as an "ordinary" predicate, entails the same set of consequences as P, where $\approx$ is taken to have the usual semantics of equality.

Now for a query predicate Q and a program P without equality, computing all $Q(\mathbf{a})$, such that $P \models_c Q(\mathbf{a})$, can be performed as follows.

Definition 6. *For a predicate symbol Q, let $\mathcal{R}_Q$ denote the standard ordered resolution calculus parameterized as follows: (*i*) all negative literals are selected, and (*ii*) all ground atoms of the form $Q(\mathbf{a})$ are smallest in the ordering $\succ$.*

Any lexicographic ordering where Q is the smallest predicate symbol is compatible with Definition 6. As shown by the following lemma, saturating P by $\mathcal{R}_Q$ computes all ground consequences involving Q. Observe that the restriction that Q does not occur in the body of some rule is not really a limitation: one can always introduce a new predicate A_Q, add the rule $A_Q(\mathbf{x}) \leftarrow Q(\mathbf{x})$, and use A_Q for query answering.

Lemma 4. *Let P be a satisfiable disjunctive datalog program and Q a predicate not occurring in the body of any rule in P. Then $P \models_c Q(\mathbf{a})$ if and only if $Q(\mathbf{a}) \in N$, where N is the set of clauses obtained by saturating P under $\mathcal{R}_Q$ up to redundancy.*

Namely, $P \models_c Q(\mathbf{a})$ if and only if the empty clause can be derived by saturating $P \cup \{\neg Q(\mathbf{a})\}$ by $\mathcal{R}_Q$. Since P is satisfiable, the empty clause is not derived by saturating P alone. Furthermore, since Q does not occur in a body of any rule in P, Q occurs negated only in $\neg Q(\mathbf{a})$. Hence, in saturating $P \cup \{\neg Q(\mathbf{a})\}$, the only inference preformed in addition to the case when P is saturated alone is a hyperresolution with $\neg Q(\mathbf{a})$. However, the literals $Q(\mathbf{a})$ are minimal, so the empty clause can be derived only if saturation of P derives $Q(\mathbf{a})$. In [18], we show that, assuming unary coding of numbers and a bound on the arity of the non-DL-predicates, saturation of $\mathsf{DD}(KB) \cup P$ can be performed in time exponential in $|KB|$, which makes our query answering algorithm worst-case optimal.

Apart from the fact that this algorithm computes all non-ground consequences related to Q in one pass, this algorithm has another interesting property. Namely, instead of performing an inference with each literal of a ground disjunction, it is sufficient to perform inferences only with the maximal literal. This dramatically reduces the number of inferences to be performed. Furthermore, if the program is not disjunctive, then hyperresolution becomes exactly the least fixpoint operator used to evaluate non-disjunctive programs; its consequences can be computed for non-disjunctive programs in polynomial time, so we get tractable behavior. In this way our algorithm supports the principle of "graceful degradation:" a performance penalty is paid only for features actually used.

Finally, we note that evaluating queries can be further optimized by using the magic sets transformation. In its basic form for Horn programs [5], magic sets transformation reduces the amount of irrelevant computation

by simulating the binding propagation of top-down SLD-resolution. This technique has recently been extended to disjunctive programs [11], and has shown significant benefits in practice. Magic sets transformation is not tied to the query evaluation method, and therefore can be used in conjunction with query answering by the $\mathcal{R}_Q$ calculus.

5.6 Example

To help in understanding the material presented, we now give a simple example. Let KB be the following knowledge base:

(15)	$\geq 2\, hasChild \sqsubseteq TaxCut$	People with ≥ 2 children get a tax cut.
(16)	$Man \sqcap Woman \sqsubseteq \bot$	Men and women are disjoint.
(17)	$\exists motherOf.\top \sqsubseteq Woman$	Mothers are women.
(18)	$\exists hasChild.(\exists motherOf.\top)(Peter)$	*Peter* has a child who is a mother.
(19)	$hasChild(Peter, Paul)$	*Peter* has a child *Paul*.
(20)	$Man(Paul)$	*Paul* is a man.

Now KB entails $TaxCut(Peter)$, and here is why: the unnamed child of *Peter* implied by (18) is a mother of someone, so this unnamed child of *Peter* must be a woman by (17). Furthermore, this unnamed child must be some other child than *Paul*, since *Paul* is a man. Hence, *Peter* has at least two children, so he is eligible for a tax cut. Let us now show how this conclusion can be drawn by reducing the knowledge base to disjunctive datalog. First, (18) contains a non-atomic concept $\exists hasChild.(\exists motherOf.\top)$, so we replace it with a new atomic concept Q_1. Furthermore, $\exists hasChild.(\exists motherOf.\top)$ contains a subconcept $\exists motherOf.\top$, so we introduce a new concept Q_2 for it. Hence, (18) is replaced with these axioms:

(21)	$Q_1(Peter)$
(22)	$Q_1 \sqsubseteq \exists hasChild.Q_2$
(23)	$Q_2 \sqsubseteq \exists motherOf.\top$

We now compute $\Xi(KB)$ by simply translating all axioms into first-order logic. We define the precedence relation $>_P$ for the LPO as follows:

$$g >_P f >_P$$
$$Peter >_P Paul >_P$$
$$Woman >_P Man >_P TaxCut >_P motherOf >_P hasChild >_P Q_2 >_P Q_1$$

Below are the closures from $\Xi(KB_{\mathcal{T}})$, where literals eligible for inferences have been underlined (they are either selected or maximal):

(24)	$TaxCut(x) \vee \underline{\neg hasChild(x, y_1)} \vee \underline{\neg hasChild(x, y_2)} \vee y_1 \approx y_2$
(25)	$\neg Man(x) \vee \underline{\neg Woman(x)}$

(26) $\underline{\neg motherOf(x,y)} \vee Woman(x)$

(27) $\neg Q_1(x) \vee \underline{hasChild(x, f(x))}$

(28) $\neg Q_1(x) \vee \underline{Q_2(f(x))}$

(29) $\neg Q_2(x) \vee \underline{motherOf(x, g(x))}$

Now we apply the inference rules of the basic superposition calculus to saturate the TBox (we show only non-redundant consequences):

(30) $\neg Q_2(x) \vee \underline{Woman(x)}$ resolve (26) and (29)

(31) $\neg Q_2(x) \vee \underline{\neg Man(x)}$ resolve (30) and (25)

We now eliminate function symbols from the saturated set by applying the operator λ to obtain the following datalog program:

(32) $TaxCut(x) \vee y_1 \approx y_2 \leftarrow hasChild(x, y_1), hasChild(x, y_2)$ $= \lambda(24)$

(33) $\leftarrow Man(x), Woman(x)$ $= \lambda(25)$

(34) $Woman(x) \leftarrow motherOf(x, y)$ $= \lambda(26)$

(35) $hasChild(x, x_f) \leftarrow Q_1(x), S_f(x, x_f)$ $= \lambda(27)$

(36) $Q_2(x_f) \leftarrow Q_1(x), S_f(x, x_f)$ $= \lambda(28)$

(37) $motherOf(x, x_g) \leftarrow Q_2(x), S_g(x, x_g)$ $= \lambda(29)$

(38) $Woman(x) \leftarrow Q_2(x)$ $= \lambda(30)$

(39) $\leftarrow Man(x), Q_2(x)$ $= \lambda(31)$

Since there are no unsafe rules, there is no need to append the Herbrand universe declarations. We merely append the ABox clauses and the definitions of S_f and S_g:

(40) $Q_1(Peter)$

(41) $hasChild(Peter, Paul)$

(42) $Man(Paul)$

(43) $S_f(Peter, Peter_f)$

(44) $S_f(Paul, Paul_f)$

(45) $S_g(Peter, Peter_g)$

(46) $S_g(Paul, Paul_g)$

To answer the query $?-TaxCut(\mathtt{x})$, we apply the algorithm from Section 5.5, where all literals involving the predicate *TaxCut* are smallest. As a result, in (52) we indeed deduce that Peter gets a tax cut.

(47) $hasChild(Peter, Peter_f)$ resolve (35)+(40)+(43)

(48) $Peter_f \approx Paul \vee TaxCut(Peter)$ resolve (32)+(41)+(47)

(49)	$S_f(Peter, Paul) \vee TaxCut(Peter)$	superpose (48) into (43)
(50)	$Q_2(Paul) \vee TaxCut(Peter)$	resolve (36)+(40)+(49)
(51)	$Woman(Paul) \vee TaxCut(Peter)$	resolve (38)+(50)
(52)	$TaxCut(Peter)$	resolve (33)+(42)+(51)

6 Related Work

$\mathcal{AL}$-log [6] 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, we present a query answering algorithm as an extension of deductive database techniques running in deterministic exponential time.

A comprehensive study of the effects of combining datalog rules with description logics is presented in [22]. The logic considered is $\mathcal{ALCNR}$, which, although less expressive than $\mathcal{SHIQ}$, contains constructors that are characteristic of most DL languages. The results of the study can be summarized as follows: (*i*) answering conjunctive queries over $\mathcal{ALCNR}$ knowledge bases is decidable, (*ii*) query answering in the extension of $\mathcal{ALCNR}$ with non-recursive datalog rules, where both concepts and roles can occur in rule bodies (but not in rule heads), 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 Semantic Web Rule Language (SWRL) [15] 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. Hence, apart from technicalities such as allowing concept expressions to occur in the rules, the formalism is compatible with DL rules. 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. DL-safe rules are a proper subset of SWRL, where some expressivity is traded for decidability. Hence, our approach provides an optimal query answering algorithm covering a significant portion of SWRL.

In [8] an approach for combining answer set programming with description logics is presented. The interaction between the subsystems is enabled by exchanging only unit (i.e. non-disjunctive) ground consequences between the two components. The set of derivable facts is obtained by fixpoint computation. In this approach, the two systems are not tightly integrated since interaction between the systems is performed only through the exchange of consequences. As a consequence, the resulting semantics is incompatible with the first-order semantics; for example, the fact that *Oedipus* is a child cannot be derived from the fact that he is a *GoodChild* or a *BadChild*′, c.f. our example at the end of Section 4.3.

The approaches from [12] and [30] for reducing certain fragments of description logics to logic programming can easily be extended with rules, by simply appending the rules to the result of the transformation. However, the description logic considered there does not support existential quantifiers, negation, or disjunction under positive polarity, so it is significantly less expressive than $\mathcal{SHIQ}(\mathbf{D})$. Hence, our approach is a proper extension.

7 Summary and Outlook

We have presented an approach for extending OWL-DL with DL-safe rules which yields a logic with decidable reasoning algorithms. 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 a practical algorithm for answering queries over $\mathcal{SHIQ}$ extended with DL-safe rules. This algorithm transforms a $\mathcal{SHIQ}$ knowledge base into a disjunctive program. To attenuate the increased computational complexity introduced by using disjunctive programs, 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. Due to space constraints, in this paper we have only presented an algorithm for $\mathcal{SHIQ}$ knowledge bases; we present an extension to $\mathcal{SHIQ}(\mathbf{D})$

in [18]. Since this algorithm handles OWL-DL apart from nominals, we believe it provides a good foundation for extending Semantic Web ontology languages with rules.

In our future work, we shall investigate how 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,[3] for which we shall conduct a thorough performance evaluation.

Acknowledgements

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. IJCAI-91*, pages 452–457, 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] C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. PODS-87*, pages 269–293, March 1987.

[6] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[7] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[8] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Proc. KR2004*. AAAI Press, 2004.

[9] M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Springer Verlag, 1996.

[3] http://kaon2.semanticweb.org/

[10] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *Proc. LICS-97*, 1997.

[11] S. Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):717–736, March/April 2003.

[12] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. WWW-2003*, pages 48–57. ACM, 2003.

[13] V. Haarslev and R. Möller. RACER System Description. In *Proc. IJCAR-01*, pages 701–706. Springer-Verlag, 2001.

[14] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. KR-98*, pages 636–647. Morgan Kaufmann Publishers, 1998.

[15] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. WWW-2004*. ACM, 2004.

[16] I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In *Proc. IJCAI-2001*, pages 199–204. Morgan Kaufmann, 2001.

[17] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[18] 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.

[19] U. Hustadt, B. Motik, and U. Sattler. Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution. In *Proc. ECAI-2004*, pages 353–357, August 2004.

[20] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proc. KR-2004*, pages 152–162. AAAI Press, June 2004.

[21] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. A Decomposition Rule for Decision Procedures by Resolution-based Calculi. In *Proc. LPAR 2004*, volume 3452, pages 21–35. Springer, March 2005.

[22] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[23] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *Proc. ISWC-2004*, pages 549–563. Springer, November 2004.

[24] R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Logic and Computation*, 19(4):312–351, April 1995.

[25] 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.

[26] D. A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Transformation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.

[27] O. Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[28] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis, RWTH Aachen, Germany, 2001.

[29] M. Vardi. Why is modal logic so robustly decidable? volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. AMS, 1997.

[30] R. Volz. *Web Ontology Reasoning With Logic Databases.* PhD thesis, Universität Fridericiana zu Karlsruhe (TH), Germany, 2004.

Information about Authors

Boris Motik is a research assistant working in the Knowledge Management (WIM) research department at the FZI Research Center for Information Technologies at the University of Karlsruhe. His research interests include description and non-classical logics, automated theorem proving, deductive databases, non-monotonic logics, and their applications in the Semantic Web. He has worked in industry for six years on projects in fields ranging from e-business applications to multimedia systems. He received a BS degree in electrical engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia in 1996 and an MS degree in computer science from the same faculty in 1999.

Ulrike Sattler is a Senior Lecturer in the School of Computer Science of the University of Manchester. Her research interests are in logics for knowledge representation and automated deduction. More specifically, she is interested in description, modal, and dynamic logics, their use in knowledge representation and as logical underpinning of ontology languages, the corresponding inference problems, their complexity, and decision procedures for these problems. She obtained her PhD from the University of Technology in Aachen and her habilitation from the Technical University in Dresden, both while working in the group of Franz Baader.

Rudi Studer is a full professor in applied informatics at AIFB Institute at the University of Karlsruhe, the director of the Knowledge Management (WIM) research department at the FZI Research Center for Information Technologies at the University of Karlsruhe, and a co-founder of the spin-off company Ontoprise GmbH, Karlsruhe. His research interests include knowledge management, Semantic Web technologies and applications, ontology engineering, knowledge discovery and peer-to-peer systems. He was awarded a PhD degree in mathematics and computer science at the University of Stuttgart, and obtained his habilitation in computer science from the same university. He worked as a research scientist from 1977 to 1985 at the University of Stuttgart. From 1985 to 1989 he was project leader and manager at the Scientific Center of IBM Germany. He is a member of AAAI, ACM, IEEE, and GI.


Boris Motik


Ulrike Sattler


Rudi Studer