Resolution-Based Reasoning for Ontologies

Boris Motik

School of Computer Science,
University of Manchester,
Manchester, UK
bmotik@cs.man.ac.uk

Summary. We overview the algorithms for reasoning with description logic (DL) ontologies based on resolution. These algorithms often have worst-case optimal complexity, and, by relying on vast experience in building resolution theorem provers, they can be implemented efficiently. Furthermore, we present a resolution-based algorithm that reduces a DL knowledge base into a disjunctive datalog program, while preserving the set of entailed facts. This reduction enables the application of optimization techniques from deductive databases, such as magic sets, to reasoning in DLs. This approach has proven itself in practice on ontologies with relatively small and simple TBoxes, but large ABoxes.

1 Introduction

Tableau algorithms, introduced in Chapter 23, are nowadays the state-of-the-art for reasoning with description logic (DL) ontologies. This is mainly due to optimizations of the original algorithm that heuristically guide the search for a model. DLs such as the ones underlying the Web Ontology Language (OWL) (see Chapter 4) are, however, complex logics, so no one reasoning method can be identified as the best. Rather, comparing different methods and identifying which ones are suitable for which types of problems can give us crucial insights into building practical reasoning systems. Therefore, alternatives to tableau calculi have been explored in the past.

Resolution and its refinements [4] are nowadays the most widely used calculi for general-purpose first-order theorem proving. They have been implemented in a number of practical systems, of which Vampire [28] is one of the most successful one. The general applicability of resolution is partly due to the powerful *redundancy elimination rules*, which can drastically reduce the search space.

Since resolution has been quite successful as a general theorem proving technique, it is natural to apply it to ontology reasoning. Decision procedures for various DLs have been developed in the past. It turns out that, even for relatively complex DLs, resolution-based algorithms can be derived easily and are quite elegant. While tableau algorithms need sophisticated blocking techniques to ensure termination [8],

resolution-based algorithms terminate automatically as a side-effect of the resolution calculus. Furthermore, many resolution-based procedures are worst-case optimal [25, 13].

In this chapter, we outline the principles underlying most known resolution-based procedures for DLs. After introducing the basic notions in Section 2, we present a decision procedure for the DL $\mathcal{ALCHI}$ in Section 3. This DL provides many features characteristic of the DL languages, such as full Boolean connectives, (restricted) existential and universal quantification, inverse roles, and role hierarchies. Furthermore, the resolution decision procedure for this DL conveys the basic principles without overloading the presentation with technical detail. We also overview the problems involved in extending the algorithm to more expressive DLs.

Deductive databases have been successfully applied to answering queries over large data sets, so it is natural to apply them to DL reasoning with large ABoxes. To enable this, in Section 4 we present a technique that reduces an $\mathcal{ALCHI}$ knowledge base to a disjunctive datalog program without affecting the set of entailed ground facts. Thus, one can answer DL queries using the resulting disjunctive program, and, in doing so, one can apply known optimization techniques such as magic sets [6]. This transformation can be derived easily from the basic resolution-based decision algorithm.

The techniques presented in this chapter have been implemented in the DL reasoner KAON2.[1] Practical experience has shown that the reduction-based techniques work quite well for ontologies with relatively small and simple TBoxes, but large and complex ABoxes [23].

2 Preliminaries

2.1 The Description Logic $\mathcal{ALCHI}$

Description logics have been introduced in detail in Chapter 1, but, to make this chapter self-contained, we present the definition of the DL $\mathcal{ALCHI}$. For a set of role names N_R, a *role* is either some $R \in N_R$ or an *inverse role* R^- for $R \in N_R$. An *RBox* $\mathcal{R}$ is a finite set of role inclusion axioms $R \sqsubseteq S$. For a set of *concept names* N_C, the set of *concepts* is the smallest set containing $\top$, $\bot$, A, $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, where A is a concept name, C and D are concepts, and R is a role. A TBox $\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$, where C and D are concepts. For a set of *individuals* N_I, an ABox $\mathcal{A}$ is a finite set of assertions of the form $C(a)$, $R(a, b)$, and $\neg R(a, b)$, where C is a concept, R is a role, and a and b are individuals. An $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$. With $|\mathcal{K}|$ we denote the number of symbols needed to encode $\mathcal{K}$. We say that $\mathcal{K}$ is *extensionally reduced* if, in all ABox assertions $C(a)$, the concept C is a concept name or the negation of a concept name. Any $\mathcal{K}$ can be made extensionally reduced by replacing each assertion $C(a)$ where C is not of the appropriate form with an assertion $A_C(a)$ and an axiom $A_C \sqsubseteq C$, for A_C a new concept name.

[1] `http://kaon2.semanticweb.org/`

Table 1. Semantics of $\mathcal{ALCHI}$ by Mapping to FOL

Mapping Roles to FOL	
$\pi_{xy}(R) = R(x,y)$	$\pi_{yx}(R) = R(y,x)$
$\pi_{xy}(R^-) = R(y,x)$	$\pi_{yx}(R^-) = R(x,y)$
Mapping Concepts to FOL	
$\pi_x(\top) = \top$	$\pi_y(\top) = \top$
$\pi_x(\bot) = \bot$	$\pi_y(\bot) = \bot$
$\pi_x(A) = A(x)$	$\pi_y(A) = A(y)$
$\pi_x(\neg C) = \neg\pi_x(C)$	$\pi_y(\neg C) = \neg\pi_y(C)$
$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$	$\pi_y(C \sqcap D) = \pi_y(C) \wedge \pi_y(D)$
$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$	$\pi_y(C \sqcup D) = \pi_y(C) \vee \pi_y(D)$
$\pi_x(\exists R.C) = \exists y : \pi_{xy}(R) \wedge \pi_y(C)$	$\pi_y(\exists R.C) = \exists x : \pi_{yx}(R) \wedge \pi_x(C)$
$\pi_x(\forall R.C) = \forall y : \pi_{xy}(R) \rightarrow \pi_y(C)$	$\pi_y(\forall R.C) = \forall x : \pi_{yx}(R) \rightarrow \pi_x(C)$
Mapping Axioms to FOL	
$\pi(C \sqsubseteq D) = \forall x : \pi_x(C) \rightarrow \pi_x(D)$	
$\pi(R \sqsubseteq S) = \forall x, y : \pi_{xy}(R) \rightarrow \pi_{xy}(S)$	
$\pi(C(a)) = \pi_x(C)\{x \mapsto a\}$	
$\pi((\neg)R(a,b)) = (\neg)\pi_{xy}(R)\{x \mapsto a, y \mapsto b\}$	
$\pi(\mathcal{K}) = \bigwedge_{\alpha \in \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}} \pi(\alpha)$	

In Chapter 1, DLs are given a direct model-theoretic semantics. In this chapter, however, we use an equivalent semantics based on translation into first-order logic. In particular, we translate an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$ into a first-order formula $\pi(\mathcal{K})$, where π is the operator defined in Table 1. It is well-known that these two semantics are equivalent [7].

The basic inference problem for $\mathcal{ALCHI}$ is *checking satisfiability* of $\mathcal{K}$—that is, checking whether $\pi(\mathcal{K})$ is a satisfiable first-order formula. As discussed in Chapter 1, other inference problems can be reduced to knowledge base satisfiability.

The *negation-normal form* $\mathsf{nnf}(C)$ of a concept C is the concept equivalent to C in which negation occurs only in front of concept names. The concept $\mathsf{nnf}(C)$ can be computed in time polynomial in the size of C by well-known transformations [2].

2.2 The Ordered Resolution Calculus

We use the well-known definitions of constants, variables, function symbols, terms, predicates, and formulae of first-order logic [4]. An *atom* A is a formula of the form $P(t_1, \ldots, t_n)$, where P is a predicate and t_i are terms. A *literal* L is a positive atom A or a negative atom $\neg A$. A *clause* is a multiset of literals and is written as $L_1 \vee \ldots \vee L_n$. The *empty clause* is written as $\Box$. Terms and formulae that do not contain variables are called *ground*. We say that formulae φ and ψ are *equisatisfiable* if φ is satisfiable if and only if ψ is satisfiable.

A *substitution* is mapping of variables to terms that is not identity on a finite number of variables; we often write it as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. An application of a substitution σ to a term t (formula φ) is written $t\sigma$ ($\varphi\sigma$) and it is the term

(formula) obtained by replacing each free occurrence of a variable x with $x\sigma$. A substitution σ is a *unifier* of terms s and t if $s\sigma = t\sigma$. A unifier σ of s and t is called a *most general unifier* if, for each unifier η of s and t, a substitution ξ exists such that $x\eta = (x\sigma)\xi$ for every variable x. If a most general unifier σ of s and t exists, it is unique up to variable renaming [3], so we write $\sigma = \mathsf{MGU}(s, t)$.

The *skolemization* of a formula φ, written $\mathsf{sk}(\varphi)$, is obtained from φ by successively replacing each subformula $\exists x : \psi$ occurring positively or a subormula $\forall x : \psi$ occurring negatively in φ with a formula $\psi\{x \mapsto f(x_1, \ldots, x_n)\}$, where f is a new function symbol and $x_1, \ldots, x_n$ are the free variables of ψ different from x. It is well-known that φ and $\mathsf{sk}(\varphi)$ are equisatisfiable [26]. Finally, $\mathsf{Cls}(\varphi)$ is the set of clauses that is equisatisfiable with φ and is obtained by transforming $\mathsf{sk}(\varphi)$ into conjunctive normal form using the well-known transformations.

Ordered resolution [4] is a calculus that can be used to prove that a formula φ is unsatisfiable. Ordered resolution is a clausal calculus, so it cannot be applied to φ directly. First, one must compute $\mathsf{Cls}(\varphi)$. Next, one must fix the calculus' parameters. The first parameter is an *admissible* ordering on literals $\succ$—that is, an ordering that is (*i*) well-founded, stable under substitutions (i.e., $L_1 \succ L_2$ implies $L_1\sigma \succ L_2\sigma$ for all literals L_1 and L_2 and each substitution σ), and total on ground literals; (*ii*) $\neg A \succ A$ for all ground atoms A; and (*iii*) $B \succ A$ implies $B \succ \neg A$ for all atoms A and B. A literal L is maximal w.r.t. a clause C if there is no literal $L' \in C$ such that $L' \succ L$, and L is strictly maximal w.r.t. C if there is no $L' \in C$ such that $L' \succeq L$. The second parameter is a *selection function*, which assigns to each clause C a possibly empty subset of negative literals of C.

An *inference rule* is a template that specifies how a conclusion is derived given a set of premises; an *inference* is an application of an inference rule to concrete premises. With $\mathcal{R}$ we denote the ordered resolution calculus, consisting of the following inference rules, where the clauses $C \vee A \vee B$ and $D \vee \neg B$ are called the *main premises*, $C \vee A$ is called the *side premise*, and $C\sigma \vee A\sigma$ and $C\sigma \vee D\sigma$ are called *conclusions* (as usual in resolution theorem proving, we make a technical assumption that the premises do not have variables in common):

Positive factoring:
$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where (*i*) $\sigma = \mathsf{MGU}(A, B)$, (*ii*) $A\sigma$ is maximal with respect to $C\sigma \vee B\sigma$ and no literal is selected in $C\sigma \vee A\sigma \vee B\sigma$.

Ordered resolution:
$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where (*i*) $\sigma = \mathsf{MGU}(A, B)$, (*ii*) $A\sigma$ is strictly maximal with respect to $C\sigma$ and no literal is selected in $C\sigma \vee A\sigma$, (*iii*) $\neg B\sigma$ is either selected in $D\sigma \vee \neg B\sigma$, or it is maximal with respect to $D\sigma$ and no literal is selected in $D\sigma \vee \neg B\sigma$.

Ordered resolution is compatible with powerful *redundancy elimination techniques*, which allow deleting certain clauses during the theorem proving process

without loss of completeness [4]. If a clause C is redundant in some set of clauses N, then C can be safely removed from N.

If a clause C is a tautology, then it is redundant in any set of clauses N. A sound and complete tautology check would itself require theorem proving, and would therefore be difficult to realize. Therefore, one usually only checks for *syntactic tautologies*—that is, clauses containing the literals A and $\neg A$. A clause C *subsumes* a clause D if there is a substitution σ such that $C\sigma \subseteq D$ and $|C| < |D|$. If a clause C is subsumed by a clause from a set of clauses N, then C is redundant in N.

A *derivation* by $\mathcal{R}$ from a set of clauses N is a sequence of sets of clauses $N_0, N_1, \ldots$ such that $N_0 = N$ and, for $i \geq 0$, either (*i*) $N_{i+1} = N_i \cup \{C\}$ where C is the conclusion of an inference by $\mathcal{R}$ from premises in N_i, or (*ii*) $N_{i+1} = N_i \setminus \{C\}$ where C is redundant in N_i. Each derivation must be *fair* [4]; intuitively, this means that each applicable inference is performed after a finite number of steps. Ordered resolution is sound and complete [4]: if $\square \in N_i$ where N_i is derived by $\mathcal{R}$ from a set of clauses N_0, then N_0 is unsatisfiable; conversely, if N_0 is unsatisfiable, then, for each fair derivation by $\mathcal{R}$ from N_0, an integer i exists such that $\square \in N_i$. The process of computing a derivation by $\mathcal{R}$ from N_0 is called a *saturation* of N_0 by $\mathcal{R}$.

2.3 Disjunctive Datalog

We recapitulate the basic notions of disjunctive datalog [11]. A *datalog term* is a constant or a variable, and a *datalog atom* has the form $A(t_1, \ldots, t_n)$ or $t_1 \approx t_2$, where t_i are datalog terms. A *disjunctive datalog program with equality* P is a finite set of rules of the form $A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$ where A_i and B_j are datalog atoms. The literals A_i are called *head literals*, whereas the literals B_i are called *body literals*. Each rule is required to be *safe*—that is, each variable occurring in the rule must occur in at least one body atom. A *fact* is a rule with $m = 0$. For the semantics, we take a rule to be equivalent to a clause $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$. We consider only Herbrand models, and say that a model M of P is *minimal* if there is no model M' of P such that $M' \subsetneq M$. A ground literal A is a *cautious answer* of P (written $P \models_c A$) if A is true in all minimal models of P. First-order entailment coincides with cautious entailment for positive ground atoms.

3 Deciding Satisfiability of $\mathcal{ALCHI}$ by Resolution

The fundamental principles for deciding a first-order fragment $\mathcal{L}$ by resolution have been established by Joyner [17]. First, one selects a sound and complete clausal calculus $\mathcal{C}$. Second, one identifies the set of clauses $\mathcal{N}_\mathcal{L}$ such that (*i*) $\mathcal{N}_\mathcal{L}$ is finite for a finite signature and (*ii*) the translation of each formula $\varphi \in \mathcal{L}$ into clauses produces only clauses from $\mathcal{N}_\mathcal{L}$. Third, one demonstrates that $\mathcal{N}_\mathcal{L}$ is *closed* under $\mathcal{C}$; that is, one shows that applying an inference of $\mathcal{C}$ to clauses from $\mathcal{N}_\mathcal{L}$ produces a clause in $\mathcal{N}_\mathcal{L}$. This is sufficient to obtain a refutation decision procedure for $\mathcal{L}$: given any formula $\varphi \in \mathcal{L}$, a saturation by $\mathcal{C}$ of the clauses corresponding to φ will, in the worst case, derive all clauses of $\mathcal{N}_\mathcal{L}$. In this section, we apply these principles to obtain a procedure for checking satisfiability of an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$.

Table 2. Structural Transformation of $\mathcal{K}$

$\Theta(\mathcal{K})$	$=$	$\bigcup_{\alpha \in \mathcal{R} \cup \mathcal{A}} \Theta(\alpha) \cup \bigcup_{C_1 \sqsubseteq C_2 \in \mathcal{T}} \Theta(\top \sqsubseteq \mathsf{nnf}(\neg C_1 \sqcup C_2))$
$\Theta(A \sqsubseteq B)$	$=$	$\{A \sqsubseteq B\}$
$\Theta(A \sqsubseteq \neg B)$	$=$	$\{A \sqsubseteq \neg B\}$
$\Theta(A \sqsubseteq C_1 \sqcap C_2)$	$=$	$\Theta(A \sqsubseteq C_1) \cup \Theta(A \sqsubseteq C_2)$
$\Theta(A \sqsubseteq C_1 \sqcup C_2)$	$=$	$\{A \sqsubseteq Q_{C_1} \sqcup Q_{C_2}\} \cup \Theta(Q_{C_1} \sqsubseteq C_1) \cup \Theta(Q_{C_2} \sqsubseteq C_2)$
$\Theta(A \sqsubseteq \exists R.C)$	$=$	$\{A \sqsubseteq \exists R.Q_C\} \cup \Theta(Q_C \sqsubseteq C)$
$\Theta(A \sqsubseteq \forall R.C)$	$=$	$\{A \sqsubseteq \forall R.Q_C\} \cup \Theta(Q_C \sqsubseteq C)$
$\Theta(R \sqsubseteq S)$	$=$	$\{R \sqsubseteq S\}$
$\Theta(C(a))$	$=$	$\{Q_C(a)\} \cup \Theta(Q_C \sqsubseteq C)$
$\Theta((\neg)R(a,b))$	$=$	$\{(\neg)R(a,b)\}$

Note: A and B are concept names or $\top$; C, C_1, and C_2 are arbitrary concepts; R and S are roles; and Q_X is a new concept name not occurring in $\mathcal{K}$ that is unique for X.

3.1 Translating the Knowledge Base into Clauses

The first step in deciding satisfiability of $\mathcal{K}$ is to transform $\mathcal{K}$ into an equisatisfiable set of clauses $\Xi(\mathcal{K})$. A straightforward way of doing so is to compute $\mathsf{Cls}(\pi(\mathcal{K}))$. Such an approach, however, has two important drawbacks. First, the size of the resulting clause set could be exponential in the size of $\pi(\mathcal{K})$, due to nesting of $\sqcap$ and $\sqcup$. Second, we should exploit the structure of the formula $\pi(\mathcal{K})$ in our algorithm, but $\mathsf{Cls}(\pi(\mathcal{K}))$ does not reflect this structure. To avoid these problems, we preprocess $\mathcal{K}$ using the *structural transformation* [26, 27].

Definition 1. *For an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$, the knowledge base $\Theta(\mathcal{K})$ is computed as shown in Table 2.*

Intuitively, this transformation replaces complex concepts with simpler ones. The knowledge base $\Theta(\mathcal{K})$ does not contain $\sqcap$, so it can be translated into clauses without an exponential blowup.

Lemma 1. *An $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$ and $\Theta(\mathcal{K})$ are equisatisfiable.*

Proof. Consider a single application of Θ. It is obvious that the axioms obtained after the transformation imply the axiom before the transformation, which proves the ($\Leftarrow$) direction. For the ($\Rightarrow$) direction, simply observe that each interpretation I of $\mathcal{K}$ can be extended to an interpretation I' of $\Theta(\mathcal{K})$ by interpreting each newly introduced concept Q_X as X. $\square$

To obtain a set of clauses corresponding to $\mathcal{K}$, we translate $\Theta(\mathcal{K})$ into first-order logic using the operator π from Table 1, skolemize it, and transform the result into conjunctive normal form. This is captured by the following definition:

Definition 2. *For an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$, let $\Xi(\mathcal{K}) = \mathsf{Cls}(\pi(\Theta(\mathcal{K})))$.*

We now show that clausification does not affect the satisfiability of a knowledge base, and that it produces clauses of a certain syntactic structure:

Table 3. Clause Types after Clausification

Axiom	Clause
$R \sqsubseteq S$	$\neg R(x,y) \vee S(x,y)$
$R^- \sqsubseteq S^-$	$\neg R(y,x) \vee S(y,x)$
$R \sqsubseteq S^-$	$\neg R(x,y) \vee S(y,x)$
$R^- \sqsubseteq S$	$\neg R(y,x) \vee S(x,y)$
$A \sqsubseteq \bigsqcup (\neg) B_i$	$\neg A(x) \vee \bigvee (\neg) B_i(x)$
$A \sqsubseteq \exists R.B$	$\neg A(x) \vee R(x, f(x))$
	$\neg A(x) \vee B(f(x))$
$A \sqsubseteq \exists R^-.B$	$\neg A(x) \vee R(f(x), x)$
	$\neg A(x) \vee B(f(x))$
$A \sqsubseteq \forall R.B$	$\neg A(x) \vee \neg R(x,y) \vee B(y)$
$A \sqsubseteq \forall R^-.B$	$\neg A(x) \vee \neg R(y,x) \vee B(y)$
$A(c)$	$A(c)$
$(\neg) R(c,d)$	$(\neg) R(c,d)$
$(\neg) R^-(c,d)$	$(\neg) R(d,c)$

Note: The function symbol f is different for each axiom.

Lemma 2. *The following claims hold for each $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$:*

1. *$\mathcal{K}$ is satisfiable if and only if $\Xi(\mathcal{K})$ is satisfiable.*
2. *$\Xi(\mathcal{K})$ can be computed in time polynomial in $|\mathcal{K}|$.*
3. *Each clause in $\Xi(\mathcal{K})$ is of the form as shown in Table 3.*

Proof. (1) Equisatisfiability of $\mathcal{K}$ and $\Xi(\mathcal{K})$ is a direct consequence of Lemma 1. (2) The number of recursive invocations of Θ and the number of new concepts Q_X are linear in $|\mathcal{K}|$. Hence, $|\Theta(\mathcal{K})|$ is linear in $|\mathcal{K}|$, so $|\Xi(\mathcal{K})|$ is polynomial in $|\mathcal{K}|$. (3) It is easy to see that $\Theta(\mathcal{K})$ contains only axioms from the left-hand side of Table 3, which are translated into clauses as shown on the right-hand side of the table. □

3.2 Saturation by Ordered Resolution

Since ordered resolution ($\mathcal{R}$) is a sound and complete calculus, we can use it to check satisfiability of $\Xi(\mathcal{K})$. To obtain a decision procedure, we just need to ensure that each saturation of $\Xi(\mathcal{K})$ by $\mathcal{R}$ terminates; that is, we must ensure that we can derive only finitely many clauses from $\Xi(\mathcal{K})$ by applying the rules of $\mathcal{R}$. There are two main reasons why we might derive an infinite number of clauses.

First, we might derive clauses with ever deeper terms. This is shown by the following example, in which the selected literals are underlined:

$$\dfrac{\dfrac{\underline{C(a)} \quad \underline{\neg C(x) \vee C(f(x))}}{C(f(a))} \qquad \underline{\neg C(x) \vee C(f(x))}}{C(f(f(a)))}$$

Second, we might derive clauses with an unbounded number of variables. For example, the following inference increases the number of variables by one, and repeating it for the conclusion produces clauses with an arbitrary number of variables:

$$\frac{\neg C(x) \vee \neg R(x,y) \vee \underline{C(y)} \qquad \underline{\neg C(y)} \vee \neg R(y,z) \vee C(z)}{\neg C(x) \vee \neg R(x,y) \vee \neg R(y,z) \vee C(z)}$$

The inferences that ordered resolution performs on a given set of premises are determined by the parameters of the calculus—the literal ordering and the selection function. By choosing these parameters appropriately, we can restrict the resolution inferences in a way that allows us to establish a bound on the term depth and on the number of variables. In the first example, if we ensure that $C(f(x)) \succ \neg C(x)$, then the second premise can participate in an inference only on literal $C(f(x))$; since $C(f(x))$ and $C(a)$ do not unify, no inference of $\mathcal{R}$ is applicable to $\underline{C(a)}$ and $\neg C(x) \vee \underline{C(f(x))}$. In the second example, the undesirable inference can be prevented if we select $\neg R(x,y)$.

The following definition fixes the parameters for $\mathcal{R}$ that, as we shall see shortly, restrict the inferences on $\Xi(\mathcal{K})$ in a way which ensures termination.

Definition 3. *Let $\mathcal{R}_{DL}$ denote the calculus $\mathcal{R}$ parameterized as follows:*

- *The literal ordering is any admissible ordering $\succ$ such that, for all function symbols f and predicates R, C, and D, we have $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$.*
- *The selection function selects every negative binary literal in each clause.*

An ordering compatible with Definition 3 can be obtained by instantiating a *lexicographic path ordering* [10]; see [22, Section 4.4] for details.

It is easy to see that an application of $\mathcal{R}_{DL}$ to clauses from Table 3 can produce clauses of the form not shown in the table. Therefore, we generalize Table 3 to *$\mathcal{ALCHI}$-clauses*, shown in Table 4. It is easy to see that $\Xi(\mathcal{K})$ contains only $\mathcal{ALCHI}$-clauses. As we show next, when applied to $\mathcal{ALCHI}$-clauses, each $\mathcal{R}_{DL}$ inference produces an $\mathcal{ALCHI}$-clause.

Lemma 3. *Each $\mathcal{R}_{DL}$ inference, when applied to $\mathcal{ALCHI}$-clauses, produces an $\mathcal{ALCHI}$-clause.*

Proof. We summarize all possible $\mathcal{R}_{DL}$ inferences on all types of $\mathcal{ALCHI}$-clauses in Table 5. For the sake of brevity, we omit inferences in which participating literals are complemented. The notation $n + m = k$ above each inference means that the inference premises are of types n and m, and the conclusion is of type k. Due to the requirement on the literal ordering $\succ$, a literal of the form $(\neg)A(x)$ occurring in a clause C can participate in an inference only if C does not contain a literal of the form $(\neg)B(f(x))$ or $R(x, f(x))$. Furthermore, a ground literal $A(a)$ does not unify with a literal $A(f(x))$, and $R(a, b)$ does not unify with $R(x, f(x))$. Hence, ground clauses can participate only in inferences with clauses not containing terms of the form $f(x)$. One can easily see that the conclusion is always an $\mathcal{ALCHI}$ clause. □

Table 4. Types of $\mathcal{ALCHI}$-Clauses

1	$\neg R(x,y) \vee S(x,y)$
2	$\neg R(x,y) \vee S(y,x)$
3	$\mathbf{P}(x) \vee R(x,f(x))$
4	$\mathbf{P}(x) \vee R(f(x),x)$
5	$\mathbf{P_1}(x) \vee \mathbf{P_2}(f(x))$
6	$\mathbf{P_1}(x) \vee \neg R(x,y) \vee \mathbf{P_2}(y)$
7	$\mathbf{P}(\mathbf{a})$
8	$(\neg)R(a,b)$

Note: $\mathbf{P}(t)$ is a possibly empty disjunction of the form $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$ for t a term of the form x, $f(x)$, or a; $\mathbf{P}(\mathbf{a})$ is a possibly empty disjunction of the form $P_1(a_1) \vee \ldots \vee P_m(a_m)$; and the empty clause $\Box$ is of type 5.

The following lemma shows that the number of $\mathcal{ALCHI}$-clauses is finite for a finite knowledge base $\mathcal{K}$. In fact, the bound on the number of derivable clauses can be used to estimate the complexity of the algorithm.

Lemma 4. *For an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$, the longest $\mathcal{ALCHI}$-clause over the signature of $\Xi(\mathcal{K})$ is polynomial in $|\mathcal{K}|$, and the number of such clauses different up to variable renaming is exponential in $|\mathcal{K}|$.*

Proof. The number c of unary predicates in the signature of $\Xi(\mathcal{K})$ is linear in $|\mathcal{K}|$, since each concept introduced by Θ corresponds to one nonliteral subconcept of C. Similarly, the number f of unary function symbols in the signature of $\Xi(\mathcal{K})$ is linear in $|\mathcal{K}|$, since each function symbol is introduced by skolemizing one concept of the form $\exists R.C$. Consider now the longest $\mathcal{ALCHI}$-clause Cl_6 of type 6. Such a clause contains a possibly negated literal $A(x)$ for each unary predicate A, and a possibly negated literal $A(f(x))$ for each pair of unary predicate and function symbols, yielding at most $\ell = 2c + 2cf$ literals, which is polynomial in $|\mathcal{K}|$. Each $\mathcal{ALCHI}$-clause of type 2 is a subset of Cl_6, so there are 2^ℓ such clauses; that is, the number of clauses is exponential in $|\mathcal{K}|$. For other $\mathcal{ALCHI}$-clause types, the bounds on the length and on the number of clauses can be derived in an analogous way. $\Box$

We now state the main result of this section:

Theorem 1. *For an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$, saturating $\Xi(\mathcal{K})$ by $\mathcal{R}_{DL}$ decides satisfiability of $\mathcal{K}$ and runs in time that is at most exponential in $|\mathcal{K}|$.*

Proof. By Lemma 4, the number of clauses derivable by $\mathcal{R}_{DL}$ from $\Xi(\mathcal{K})$ is exponential in $|\mathcal{K}|$. Each inference can be performed in time polynomial in the size of clauses. Hence, the saturation terminates after performing at most an exponential number of steps. Since $\mathcal{R}_{DL}$ is sound and complete, it decides satisfiability of $\Xi(\mathcal{K})$, and by Lemma 2 of $\mathcal{K}$ as well, in time that is exponential in $|\mathcal{K}|$. $\Box$

Table 5. Possible Inferences by $\mathcal{R}_{DL}$ on $\mathcal{ALCHI}$-Clauses

1+3=3:

$$\frac{\underline{\neg R(x,y)} \vee S(x,y) \quad \mathbf{P}(x) \vee \underline{R(x,f(x))}}{\mathbf{P}(x) \vee S(x,f(x))}$$

2+3=4:

$$\frac{\underline{\neg R(x,y)} \vee S(y,x) \quad \mathbf{P}(x) \vee \underline{R(x,f(x))}}{\mathbf{P}(x) \vee S(f(x),x)}$$

1+4=4:

$$\frac{\underline{\neg R(x,y)} \vee S(x,y) \quad \mathbf{P}(x) \vee \underline{R(f(x),x)}}{\mathbf{P}(x) \vee S(f(x),x)}$$

2+4=3:

$$\frac{\underline{\neg R(x,y)} \vee S(y,x) \quad \mathbf{P}(x) \vee \underline{R(f(x),x)}}{\mathbf{P}(x) \vee S(x,f(x))}$$

6+3=5:

$$\frac{\mathbf{P_1}(x) \vee \underline{\neg R(x,y)} \vee \mathbf{P_2}(y) \quad \mathbf{P}(x) \vee \underline{R(x,f(x))}}{\mathbf{P}(x) \vee \mathbf{P_1}(x) \vee \mathbf{P_2}(f(x))}$$

6+4=5:

$$\frac{\mathbf{P_1}(x) \vee \underline{\neg R(x,y)} \vee \mathbf{P_2}(y) \quad \mathbf{P}(x) \vee \underline{R(f(x),x)}}{\mathbf{P}(x) \vee \mathbf{P_1}(f(x)) \vee \mathbf{P_2}(x)}$$

5+5=5:

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(f(x)) \vee \underline{\neg A(f(x))} \quad \underline{A(x)} \vee \mathbf{P_3}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(f(x)) \vee \mathbf{P_3}(f(x))}$$

5+5=5:

$$\frac{\mathbf{P_1}(x) \vee \underline{\neg A(x)} \quad \underline{A(x)} \vee \mathbf{P_2}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x)}$$

5+5=5:

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(f(x)) \vee \underline{\neg A(f(x))} \quad \underline{A(f(x))} \vee \mathbf{P_3}(f(x)) \vee \mathbf{P_4}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(f(x)) \vee \mathbf{P_3}(f(x)) \vee \mathbf{P_4}(x)}$$

7+5=7:

$$\frac{\mathbf{P_1}(\mathbf{a}) \vee \underline{\neg A(b)} \quad \underline{A(x)} \vee \mathbf{P_2}(x)}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(b)}$$

7+7=7:

$$\frac{\mathbf{P_1}(\mathbf{a}) \vee \underline{\neg A(b)} \quad \underline{A(b)} \vee \mathbf{P_2}(\mathbf{c})}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(\mathbf{c})}$$

8+1=8:

$$\frac{\underline{R(a,b)} \quad \underline{\neg R(x,y)} \vee S(x,y)}{S(a,b)}$$

8+2=8:

$$\frac{\underline{R(a,b)} \quad \underline{\neg R(x,y)} \vee S(y,x)}{S(b,a)}$$

8+6=7:

$$\frac{\underline{R(a,b)} \quad \mathbf{P_1}(x) \vee \underline{\neg R(x,y)} \vee \mathbf{P_2}(y)}{\mathbf{P_1}(a) \vee \mathbf{P_2}(b)}$$

8+8=5:

$$\frac{\underline{R(a,b)} \quad \underline{\neg R(a,b)}}{\square}$$

3.3 An Example

We now present a simple example. Let $\mathcal{K}$ be the following knowledge base:

(1) $\exists S.A \sqsubseteq \exists R.B$

(2) $B \sqsubseteq C$

(3) $\exists R.C \sqsubseteq D$

(4) $S(a,b)$

(5) $A(b)$

Let us assume that we want to check whether $\mathcal{K} \models D(a)$; as shown in Chapter 1, this so if and only if $\mathcal{K} \cup \{\neg D(a)\}$ is unsatisfiable. Hence, let $\mathcal{K}'$ be the knowledge base $\mathcal{K}$ extended with the assertion $\neg D(a)$.

To check satisfiability of $\mathcal{K}'$ using resolution, we first apply structural transformation. For (1), we obtain the following:

$$\Theta(\top \sqsubseteq \forall S.\neg A \sqcup \exists R.B) = \{\top \sqsubseteq Q_1 \sqcup Q_2\} \cup \Theta(Q_1 \sqsubseteq \forall S.\neg A) \cup \Theta(Q_2 \sqsubseteq \exists R.B)$$

By Definition (1), we should introduce a new name for the concepts $\neg A$ and B; however, both $Q_1 \sqsubseteq \forall S.\neg A$ and $Q_2 \sqsubseteq \exists R.B$ can be translated into $\mathcal{ALCHI}$-clauses in a straightforward way. Hence, we do not further apply Θ, and neither we do so for (2) and (3). We obtain the set $\Xi(\mathcal{K}')$ as follows (the meaning of underlining will be explained shortly):

(6) $\top \sqsubseteq Q_1 \sqcup Q_2 \quad \rightsquigarrow \quad Q_1(x) \vee \underline{Q_2(x)}$

(7) $Q_1 \sqsubseteq \forall S.\neg A \quad \rightsquigarrow \quad \neg Q_1(x) \vee \underline{\neg S(x,y)} \vee \neg A(y)$

(8) $Q_2 \sqsubseteq \exists R.B \quad \rightsquigarrow \quad \neg Q_2(x) \vee \underline{R(x,f(x))}$

(9) $Q_2 \sqsubseteq \exists R.B \quad \rightsquigarrow \quad \neg Q_2(x) \vee \underline{B(f(x))}$

(10) $B \sqsubseteq C \quad \rightsquigarrow \quad \neg B(x) \vee \underline{C(x)}$

(11) $\exists R.C \sqsubseteq D \quad \rightsquigarrow \quad D(x) \vee \underline{\neg R(x,y)} \vee \neg C(y)$

(12) $S(a,b) \quad \rightsquigarrow \quad \underline{S(a,b)}$

(13) $A(b) \quad \rightsquigarrow \quad \underline{A(b)}$

(14) $\neg D(a) \quad \rightsquigarrow \quad \underline{\neg D(a)}$

To saturate $\Xi(\mathcal{K}')$ by $\mathcal{R}_{DL}$, we use a literal ordering $\succ$ compatible with Definition 3, where we break ties by comparing predicates alphabetically. The literals that are either selected or maximal are underlined. We now saturate $\Xi(\mathcal{K}')$; R(xx+yy) means that a clause was obtained by resolving (xx) and (yy).

(15) $D(x) \vee \neg Q_2(x) \vee \underline{\neg C(f(x))}$ R(8+11)

(16) $D(x) \vee \neg Q_2(x) \vee \underline{\neg B(f(x))}$ R(15+10)

(17) $D(x) \vee \underline{\neg Q_2(x)}$ R(16+9)

(18) $D(x) \vee \underline{Q_1(x)}$ R(17+6)

(19) $\underline{\neg Q_1(a)} \vee \neg A(b)$ R(7+12)

(20) $\underline{D(a)} \vee \neg A(b)$ R(18+19)

(21) $\underline{\neg A(b)}$ R(14+20)

(22) $\square$ R(13+21)

We derived the empty clause, so the set of clauses $\Xi(\mathcal{K}')$ is unsatisfiable, and so is $\mathcal{K}'$, which implies $\mathcal{K} \models D(a)$.

3.4 Extending the Algorithm to the More Expressive DLs

We now overview the problems encountered in extending this basic algorithm to more expressive DLs and point to the relevant literature for the solutions.

Boolean Role Expressions

The DL $\mathcal{ALB}$ [25] is obtained from $\mathcal{ALCHI}$ by allowing for concepts $\forall E.C$ and $\exists E.C$ and axioms $E_1 \sqsubseteq E_2$, where $E_{(i)}$ are *Boolean role expressions* R, $\neg E$, $E_1 \sqcup E_2$, and $E_1 \sqcap E_2$. As shown in [25], $\mathcal{ALB}$ can easily be decided by extending the algorithm from this section. The main difference is that translating an $\mathcal{ALB}$ knowledge base to clauses can produce clauses of the following form:

(23) $$\neg R_1(x,y) \vee \ldots \vee \neg R_n(x,y) \vee S_1(x,y) \vee \ldots \vee S_m(x,y)$$

If $n = 0$, such clauses can cause termination problems. For example, resolving the clauses (24) and (25) produces the clause (26):

(24) $\underline{R(x,y)}$

(25) $A(x) \vee \underline{\neg R(x,y)} \vee B(y)$

(26) $A(x) \vee \underline{B(y)}$

The clause (26) contains two clauses of type 6 that do not share a variable. Resolving such clauses with other clauses of that form can easily produce clauses with an arbitrary number of variables. For example, resolving (26) with (27) produces (28), which contains more variables than either of the premises:

(27) $\underline{\neg B(y)} \vee C(y) \vee D(z)$

(28) $A(x) \vee C(y) \vee D(z)$

This problem, however, can be solved in a simple way: since $A(x)$ and $B(y)$ are variable-disjoint, similarly as in the DPLL procedure [9], we can *split* the clause (26) into $A(x)$ or $B(y)$—that is, we can guess which subclause is true. This reduces (26) to a clause of type 6, which does not cause termination problems. Splitting makes the procedure nondeterministic: deriving the empty clause under one of the guesses does not mean that the original clause set is unsatisfiable; rather, we must derive the empty clause under all possible guesses. Hence, such an algorithm runs in NEXPTIME. This is worst-case optimal, since $\mathcal{ALB}$ is an NEXPTIME-complete logic [21].

Transitivity Axioms

Many DLs allow roles to be declared as transitive [12]. Translation of transitivity axioms produces clauses of the following form:

$$\neg R(x,y) \vee \underline{\neg R(y,z)} \vee R(x,z) \tag{29}$$

Such clauses are difficult for resolution. For example, if we also have the clause (30), then it can be resolved with (29) to produce (31):

$$A(x') \vee \underline{R(x', f(x'))} \tag{30}$$

$$\underline{\neg R(x,x')} \vee A(x') \vee R(x, f(x')) \tag{31}$$

Clause (31) is similar to (30), but it contains two variables; hence, further resolution inferences with (31) might produce clauses with even more variables.

To prevent the increase in the number of variables, one might select the negative literal in (31). While this prevents the introduction of arbitrarily many variables, it allows the derivation of arbitrarily deep terms; for example, a resolution of (30) and (31) produces the following clause:

$$A(x) \vee R(x, f(f(x))) \tag{32}$$

There are several ways to address this problem. In [18], resolution has been extended with simplification rules that transform clauses of the form (31) and (32) into simpler clauses without affecting satisfiability.

Another solution is to replace transitivity axioms with new concept inclusion axioms that capture the effects of the transitivity axioms. Roughly speaking, a transitivity axiom $\mathsf{Trans}(S)$ is replaced with axioms $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each R with $S \sqsubseteq^* R$ and C a "relevant" concept from $\mathcal{K}$; for more details, please see [22, Section 5.2]. Similar encodings have been considered in modal logic [29] and in DLs with role conjunctions [30].

Number Restrictions

As explained in Chapter 1, many DLs provide for number restrictions $\geqslant n\, R.C$ and $\leqslant n\, R.C$. The algorithm from this section can be extended to such concepts by using the well-known translation of number restrictions into first-order logic:

$$\geqslant n\, R.C \;\rightsquigarrow\; \exists y_1, \ldots, y_n : \bigwedge_{1 \leq i \leq n+1} [R(x,y_i) \wedge C(y_i)] \wedge \bigwedge_{1 \leq i < j \leq n} y_i \not\approx y_j$$

$$\leqslant n\, R.C \;\rightsquigarrow\; \forall y_1, \ldots, y_{n+1} : \bigwedge_{1 \leq i \leq n+1} [R(x,y_i) \wedge C(y_i)] \rightarrow \bigvee_{1 \leq i < j \leq n+1} y_i \approx y_j$$

These translations employs the equality predicate $\approx$. Ordered resolution alone is not an efficient calculus for theorem proving with equality. Therefore, deciding DLs with number restrictions typically requires the application of a calculus optimized

for theorem proving with equality. *Basic superposition* [5, 24] is one such calculus, which introduces new rules that take into account the semantics of equality.

In [13], a decision procedure for the DL $\mathcal{SHIQ}^-$ (a DL obtained from $\mathcal{SHIQ}$ by imposing certain restrictions on the usage of number restrictions) based on basic superposition. In [14], this algorithm has been generalized to $\mathcal{SHIQ}$ by extending basic superposition with a *decomposition* inference rule, which simplifies certain clauses. All these procedures are worst-case optimal (i.e., they run in ExpTime) for unary coding of numbers. It is known that $\mathcal{SHIQ}$ is ExpTime-complete even for binary coding of numbers [30]; however, the assumption of unary number coding is standard in practical DL reasoning systems.

Nominals

Another common construct considered in DLs are nominals. Although such a result has not been published, it would be straightforward to extend the algorithms from [13, 14] to handle the DL $\mathcal{SHOQ}$. The combination of inverse roles and nominals, however, is rather difficult to handle. Intuitively, such a logic does not have the tree-model property. Still, in [19], basic superposition has been extended with decomposition and novel *nominal generation* rule to obtain a decision procedure for $\mathcal{SHOIQ}$. The resulting decision procedure is, however, not optimal: it runs in triple exponential time, whereas $\mathcal{SHOIQ}$ is NExpTime-complete [30].

4 Reasoning by Reduction to Logic Programming

We now present an algorithm for reducing an $\mathcal{ALCHI}$ knowledge base to a disjunctive datalog program that entails the same set of ground atoms. As discussed in [23], such a reasoning technique is particularly suitable for knowledge bases that have a rather small and simple TBox but a large ABox.

4.1 The Main Difficulty

For an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$, our goal is to derive a disjunctive datalog program $\mathsf{DD}(\mathcal{K})$ such that $\mathcal{K} \models \alpha$ if and only if $\mathsf{DD}(\mathcal{K}) \models \alpha$ for α of the form $A(a)$ or $R(a, b)$. Thus, we can use $\mathsf{DD}(\mathcal{K})$ instead of $\mathcal{K}$ for query answering, and in doing so, we can apply all optimization techniques known from deductive databases, such as magic sets [6] or join-order optimizations [1].

As shown in Table 1 and in [7], there is a close correspondence between description logics and first-order logic. Consider the following knowledge base:

$$\mathcal{K} = \{A \sqsubseteq \exists R.A, \exists R.\exists R.A \sqsubseteq B, A(a)\} \tag{33}$$

A naïve attempt to reduce $\mathcal{K}$ into disjunctive datalog is to translate $\mathcal{K}$ into a first-order formula $\pi(\mathcal{K})$, skolemize it, translate it into conjunctive normal form, and rewrite the

obtained set of clauses into rules. For $\mathcal{K}$, such an approach produces the following logic program $\mathsf{LP}(\mathcal{K})$:

$$R(x, f(x)) \leftarrow A(x) \tag{34}$$

$$A(f(x)) \leftarrow A(x) \tag{35}$$

$$B(x) \leftarrow R(x, y), R(y, z), A(z) \tag{36}$$

$$A(a) \tag{37}$$

Clearly, $\mathcal{K}$ and $\mathsf{LP}(\mathcal{K})$ entail the same set of ground facts. The program $\mathsf{LP}(\mathcal{K})$, however, contains a function symbol in a recursive rule (35). This raises the issue of how to answer queries in $\mathsf{LP}(\mathcal{K})$. Namely, well-known query evaluation techniques will not terminate on $\mathsf{LP}(\mathcal{K})$; for example, using bottom-up saturation, we shall derive $A(f(a))$, $R(a, f(a))$, $A(f(f(a)))$, $R(f(a), f(f(a)))$, $B(a)$, and so on. Obviously, such an algorithm will continue deriving ever deeper facts, and will therefore never terminate. Note that we need all previously derived facts to derive $B(a)$ from $\mathsf{LP}(\mathcal{K})$, and that we do not know a priori when all relevant ground facts have been derived, so that we might stop the saturation.

This problem could be solved by employing an appropriate cycle detection mechanism. In [16], such an approach has been used to derive a decision procedure for the DL $\mathcal{ALC}$ based on hyperresolution. Using specialized algorithms for evaluating queries in $\mathsf{LP}(\mathcal{K})$ takes us, however, away from our original goal of applying deductive database optimization techniques to description logics. In a way, such an algorithm could be viewed as an alternative notation for the tableau calculus, for which it is unclear how to apply optimization techniques such as magic sets.

To avoid potential problems with termination, our goal is to derive a true disjunctive datalog program $\mathsf{DD}(\mathcal{K})$ without function symbols. For such a program, queries can be evaluated using any standard technique; furthermore, all existing optimization techniques known from deductive databases can be applied directly. Hence, the main problem that we deal with is the elimination of function symbols from $\mathsf{LP}(\mathcal{K})$.

4.2 The Translation Algorithm

From Table 5, we see that (*i*) a ground clause cannot participate in an inference with a nonground clause containing a function symbol, and (*ii*) if one premise in an inference by $\mathcal{R}_{DL}$ is ground, the conclusion is ground as well. Hence, we can perform all inferences among nonground clauses first, after which we can simply delete all nonground clauses containing function symbols. The remaining clause set consists of clauses without function symbols, which can easily be translated into a disjunctive datalog program, by moving positive literals into rule heads and negative literals into rule bodies. A minor problem arises if the resulting rules contain unsafe variables. We deal with such clauses using a simple trick: we introduce a new predicate HU and add an assertion $HU(a)$ for each individual a; next, we append $HU(x)$ to the body of each rule in which x is an unsafe variable.

Definition 4. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be an extensionally reduced $\mathcal{ALCHI}$ knowledge base. Then, $\Gamma(\mathcal{T} \cup \mathcal{R})$ is the set of clauses obtained by*

- *saturating $\Xi(\mathcal{T} \cup \mathcal{R})$ by $\mathcal{R}_{DL}$, and then*
- *deleting all clauses containing function symbols.*

The disjunctive datalog program $\mathsf{DD}(\mathcal{K})$ is obtained from $\Gamma(\mathcal{T} \cup \mathcal{R}) \cup \Xi(\mathcal{A})$ using the following transformations:

- *each clause of the form $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$ is rewritten into a rule $A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$;*
- *if a variable x occurs in some rule only in the head, then the literal $HU(x)$ is added to the rule body; and*
- *the fact $HU(a)$ is added to the program for each constant a occurring in $\mathcal{K}$.*

If $\mathcal{K}$ is not extensionally reduced, then $\mathsf{DD}(\mathcal{K}) = \mathsf{DD}(\mathcal{K}')$, where $\mathcal{K}'$ is an extensionally reduced knowledge base obtained from $\mathcal{K}$ as explained in Section 2.1.

We now state the properties of $\mathsf{DD}(\mathcal{K})$:

Theorem 2. *The following claims hold for each $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$:*

1. *$\mathcal{K}$ is satisfiable if and only if $\mathsf{DD}(\mathcal{K})$ is satisfiable.*
2. *$\mathcal{K} \models \alpha$ if and only if $\mathsf{DD}(\mathcal{K}) \models_c \alpha$, where α is of the form $A(a)$ or $R(a, b)$ for A a concept name and R a role.*
3. *$\mathcal{K} \models C(a)$ for a complex concept C if and only if $\mathsf{DD}(\mathcal{K} \cup \{C \sqsubseteq Q\}) \models_c Q(a)$ for Q a new concept name.*
4. *The number of literals in each rule in $\mathsf{DD}(\mathcal{K})$ is at most polynomial, the number of rules in $\mathsf{DD}(\mathcal{K})$ is at most exponential, and $\mathsf{DD}(\mathcal{K})$ can be computed in time exponential in $|\mathcal{K}|$.*

Proof. (1) Table 5 shows that each inference with at least one ground premise (these are the inferences below the dashed line) always produces a ground conclusion. Hence, in saturating $\Xi(\mathcal{K})$ by $\mathcal{R}_{DL}$, we can perform all inferences among nonground clauses first. Furthermore, Table 5 also shows that ground clauses can participate in inferences only with clauses not containing function symbols. Hence, after performing all inferences among nonground clauses of $\Xi(\mathcal{K})$, we can delete all clauses with terms of the form $f(x)$.

By Definition 2, $\Xi(\mathcal{T} \cup \mathcal{R})$ is exactly the set of nonground clauses of $\Xi(\mathcal{K})$, so $\Gamma(\mathcal{T} \cup \mathcal{R})$ is exactly the set of clauses obtained by saturating the nonground part of $\Xi(\mathcal{K})$ and deleting the clauses containing function symbols. Furthermore, it is easy to see that $\Gamma(\mathcal{T} \cup \mathcal{R}) \cup \Xi(\mathcal{A})$ is satisfiable if and only if $\mathsf{DD}(\mathcal{K})$ is satisfiable. Namely, both clause sets are function-free and they differ only in that the unsafe variables in the latter set are bound using the predicate HU which enumerates the entire Herbrand universe.

(2) Simply observe that $\mathcal{K} \models \alpha$ if and only if $\mathcal{K} \cup \{\neg\alpha\}$ is unsatisfiable. The latter is the case if and only if $\mathsf{DD}(\mathcal{K} \cup \{\leftarrow \alpha\}) = \mathsf{DD}(\mathcal{K}) \cup \{\leftarrow \alpha\}$ is unsatisfiable, which is the case if and only if $\mathsf{DD}(\mathcal{K}) \models_c \alpha$.

(3) Follows in the same manner as (2).

(4) Follows immediately from Lemma 4. □

4.3 An Example

We now continue the example from Section 3.3 and compute a disjunctive datalog program $\mathsf{DD}(\mathcal{K})$. The first step in the algorithm is to compute $\Xi(\mathcal{T} \cup \mathcal{R})$; clearly, it consists of the clauses (6)–(11).

The next step is to compute $\Gamma(\mathcal{T} \cup \mathcal{R})$ by saturating $\Xi(\mathcal{T} \cup \mathcal{R})$ by $\mathcal{R}_{DL}$. This was already done in Section 3.3: the saturated set contains the clauses (6)–(11) and, additionally, (15)–(18).

The next step is to remove all clauses containing function symbols. Therefore, we remove the clauses (8), (9), (15), (16). The final step is to compute $\mathsf{DD}(\mathcal{K})$ by moving all negative literals into the body and the positive literals into the head. The clauses (6) and (18) are unsafe, so we additionally add the literals $HU(x)$ to the body of the rules.

(38) $$Q_1(x) \vee Q_2(x) \leftarrow HU(x)$$

(39) $$\leftarrow Q_1(x), S(x, y), A(y)$$

(40) $$C(x) \leftarrow B(x)$$

(41) $$D(x) \leftarrow R(x, y), C(y)$$

(42) $$D(x) \leftarrow Q_2(x)$$

(43) $$D(x) \vee Q_1(x) \leftarrow HU(x)$$

Finally, we add to $\mathsf{DD}(\mathcal{K})$ the ABox and the facts involving HU:

(44) $$S(a, b)$$

(45) $$A(b)$$

(46) $$HU(a)$$

(47) $$HU(b)$$

It is straightforward to verify that $\mathsf{DD}(\mathcal{K}) \models D(a)$, in accordance with Theorem (2).

It is instructive to compare the algorithm from this section with tableaux algorithms from Chapter 23. Tableau algorithms introduce new individuals in order to satisfy the existential quantifiers. In contrast, the programs obtained by the reduction do not represent such individuals at all. In our example, $\mathsf{DD}(\mathcal{K})$ is function-free, so the universe of the program is restricted to the constants explicitly mentioned in it. Thus, the models of $\mathcal{K}$ and $\mathsf{DD}(\mathcal{K})$ coincide only on positive ground facts, and are unrelated for the facts involving unnamed objects.

To understand why the saturation of the TBox and RBox by $\mathcal{R}_{DL}$ is necessary, consider the role of each rule in $\mathsf{DD}(\mathcal{K})$. While the axiom (2) in $\mathcal{K}$ is applicable to all individuals in a model, the rule (40) is applicable only to named individuals. The relationship between (3) and (41) is analogous. To compensate for the fact that (40) and (41) derive consequences only about named individuals, $\mathsf{DD}(\mathcal{K})$ contains the rule (42), which is produced by the saturation of $\Xi(\mathcal{T} \cup \mathcal{R})$ by $\mathcal{R}_{DL}$. This rule acts as a shortcut: instead of introducing for each x in Q_2 an R-successor y in B by (8), propagating y to C by (10), and then concluding that x is in D by (11), the rule (42) derives that all instances of Q_2 are instances of D in one step. This ensures that $\mathsf{DD}(\mathcal{K})$ and $\mathcal{K}$ entail the same set of ground facts.

4.4 Discussion

By Theorem 2, the program $\mathsf{DD}(\mathcal{K})$ is independent of the query, as long as the query is a concept name or a role. Hence, $\mathsf{DD}(\mathcal{K})$ can be computed once, and can be used to answer any query involving only concept names. If the query involves a complex concept C (even if C is a negated concept name), then query answering can be reduced to entailment of positive ground facts, by introducing a new name Q and by adding the axiom $C \sqsubseteq Q$ to the TBox. Obviously, $\mathsf{DD}(\mathcal{K} \cup \{C \sqsubseteq Q\})$ may depend on C. Namely, by saturating $\Gamma(\mathcal{T} \cup \mathcal{R})$, the reduction algorithm derives all nonground consequences of $\mathcal{K}$, and a complex query concept can introduce new nonground consequences, which should be taken into account in the reduction.

Theorem 2 allows $|\mathsf{DD}(\mathcal{K})|$ to be exponential in $|\mathcal{K}|$, which may seem discouraging. Note, however, that the number of rules depends on $|\mathcal{T} \cup \mathcal{R}|$ and not on $|\mathcal{A}|$. This is important for *data complexity* [31]—the complexity under the assumption that the TBox and RBox are fixed. Under such an assumption, $|\mathsf{DD}(\mathcal{K})|$ becomes polynomial in $|\mathcal{A}|$, which has been used in [15] to show that checking satisfiability of $\mathcal{SHIQ}$ knowledge bases is NP-complete for data complexity. Also, a *Horn fragment* of $\mathcal{SHIQ}$ has been identified that does not provide for disjunctive reasoning but exhibits polynomial data complexity. To deal with the exponential blowup in the number of rules, an optimization has been presented in [13, 22] that allows many rules to be removed from $\mathsf{DD}(\mathcal{K})$ without invalidating Theorem 2. Practical experience has shown that the number of remaining rules is typically twice the number of axioms in $\mathcal{K}$ [23].

4.5 Adding Number Restrictions

The reduction algorithm presented in [13, 22] differs from this one mainly in that it can handle knowledge bases with number restrictions. We now outline the differences between this algorithm and the one presented in this section. Namely, if $\mathcal{K}$ is an $\mathcal{ALCHI}$ knowledge base, all functional terms encountered in a saturation of $\Xi(\mathcal{K})$ by $\mathcal{R}_{DL}$ are nonground (see Table 5). This is no longer the case if $\mathcal{K}$ is an $\mathcal{ALCHIQ}$ knowledge base. Namely, the translation of number restrictions can produce clauses such as (48). To see why such clauses case problems, let us assume that some other axioms produce the clauses (49)–(50).

$$\underline{\neg R(x, y_1)} \vee \underline{\neg R(x, y_2)} \vee y_1 \approx y_2 \tag{48}$$

$$\neg C(x) \vee \underline{R(x, f(x))} \tag{49}$$

$$\underline{R(a, b)} \tag{50}$$

By resolving (48) with (49) and (50), we obtain the following clause:

$$\neg C(a) \vee f(a) \approx b \tag{51}$$

This clause differs from clauses of type 7 from Table 4 in that it contains a ground functional term. The functional terms from clauses such as (51) can participate in further inferences, so we cannot just remove all clauses with function terms.

The solution is to represent ground functional terms in clauses of the form (51) using new constants. Thus, the clause (51) is encoded as the following clause, where a_f is a new constant unique for a pair of a and f:

$$\neg C(a) \vee a_f \approx b \tag{52}$$

After saturation of TBox and RBox, the nonground clauses from the saturated set are transformed in a certain way that reflects such an encoding of the ground clauses. It is important to understand that the constants such as a_f have no deeper semantic meaning; they are just a proof-theoretic aid that allows the simulation of inferences of basic superposition in disjunctive datalog.

5 Conclusion

This chapter overviews the algorithms for reasoning in description logics by resolution. These algorithms are interesting because they are worst-case optimal, but are also suitable for practical implementation [23]. Furthermore, such algorithms can be used to reduce a DL knowledge base to a disjunctive datalog program. This allows the application of known reasoning algorithms from deductive databases to reasoning with large ABoxes. Practical experience has shown that such algorithms are quite suitable for ontologies with relatively small and simple TBoxes but large ABoxes.

A challenge for future research is to obtain a more elegant and perhaps worst-case optimal algorithm for reasoning with nominals. Namely, reasoning with nominals requires reasoning about the cardinality of sets, which is known to be difficult for resolution. Another challenge is to provide methods for dealing with transitivity and general role inclusion axioms, such as the ones available in the DL $\mathcal{SROIQ}$ [20].

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, January 2003.
3. F. Baader and W. Snyder. Unification Theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. 2001.
4. 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. 2001.
5. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.
6. C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. PODS '87*, pages 269–283, San Diego, CA, USA, 1987.
7. A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

8. M. Buchheit, F. M. Donini, and A. Schaerf. Decidable Reasoning in Terminological Knowledge Representation Systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
9. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.
10. N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. 2001.
11. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
12. I. Horrocks and U. Sattler. A Description Logic with Transitive and Inverse Roles and Role Hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
13. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proc. KR 2004*, pages 152–162, Whistler, Canada, 2004.
14. U. Hustadt, B. Motik, and U. Sattler. A Decomposition Rule for Decision Procedures by Resolution-based Calculi. In *Proc. LPAR 2004*, pages 21–35, Uruguay, 2005.
15. U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. IJCAI 2005*, pages 466–471, Edinburgh, UK, 2005.
16. U. Hustadt and R. A. Schmidt. On the Relation of Resolution and Tableaux Proof Systems for Description Logics. In *Proc. IJCAI '99*, pages 202–207, Stockhom, Sweden, 1999.
17. W. H. Joyner. Resolution Strategies as Decision Procedures. *Journal of the ACM*, 23(3):398–417, 1976.
18. Y. Kazakov and H. de Nivelle. A Resolution Decision Procedure for the Guarded Fragment with Transitive Guards. In *Proc. IJCAR 2004*, pages 122–136, Cork, Ireland, 2004.
19. Y. Kazakov and B. Motik. A Resolution-Based Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. IJCAR 2006*, pages 662–667, Seattle, WA, USA, 2006.
20. O. Kutz, I. Horrocks, and U. Sattler. The Even More Irresistible SROIQ. In *Proc. KR 2006*, pages 68–78, Lake District, UK, 2006.
21. C. Lutz and U. Sattler. The Complexity of Reasoning with Boolean Modal Logics. In *Proc. AiML 2000*, pages 329–348, Leipzig, Germany, 2001.
22. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
23. B. Motik and U. Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proc. LPAR 2006*, pages 227–241, Cambodia, 2006.
24. R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, 19(4):312–351, 1995.
25. H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.
26. 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. 2001.
27. D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.
28. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2–3):91–110, 2002.
29. R. A. Schmidt and U. Hustadt. A Principle for Incorporating Axioms into the First-Order Translation of Modal Formulae. In *Proc. CADE-19*, pages 412–426, USA, 2003.
30. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.
31. M. Vardi. The Complexity of Relational Query Languages. In *Proc. STOC '82*, pages 137–146, San Francisco, CA, USA, 1982.