Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance?

Boris Motik

Forschungszentrum Informatik
Karlsruhe, Germany
motik@fzi.de

Abstract

As applications of description logics (DLs) proliferate, efficient reasoning with large ABoxes (sets of individuals with descriptions) becomes ever more important. Motivated by the prospects of reusing optimization techniques of deductive databases, we developed a novel algorithm for reasoning in description logic, which reduces a DL knowledge base to a disjunctive datalog program without changing the set of relevant consequences. This allows to answer queries by applying optimization techniques, such as join-order optimizations or magic sets. The algorithm supports the very expressive logic $\mathcal{SHIQ}(\mathbf{D})$, so the reduction is quite technically involved. In this paper we present a simplified algorithm for the basic logic $\mathcal{ALC}$. Whereas this algorithm is much easier to understand, it is based on the same principles as the general one.

1 Introduction

In recent years description logics (DLs) have found their application in various fields of computer science, so several DL reasoners were built and applied to practical problems. The state-of-the-art systems are based on tableau algorithms [11], and they perform quite well when computing the subsumption hierarchy, mainly due to sophisticated heuristics [10].

However, new applications, such as metadata management in the Semantic Web, require efficient query answering over large ABoxes (i.e., sets of ground facts). Query answering is currently implemented by a reduction to ABox consistency checking, which can then be solved by tableau algorithms. Whereas this is quite elegant from a theoretical point of view, it does not provide good performance in practice, mainly due to the following two reasons. First, tableau-based algorithms treat all individuals separately; that is, they do not group individuals on common properties. Second, to answer a query, one

usually does not need to consider all ABox information; rather, only a small subset of the ABox usually suffices to compute the query answer. These deficiencies were already acknowledged, and certain optimization techniques were developed [9]; however, the performance of query answering is still often not satisfactory in practice.

Since techniques for reasoning in deductive databases are nowadays mature, it makes sense to examine if they can be used to improve ABox reasoning in description logics. To facilitate that, we developed a novel technique for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program without changing the set of entailed ground facts [13,14,12]. This algorithm addresses the mentioned points in the following way. First, query answering in disjunctive datalog can be implemented by manipulating individuals in sets, and applying each inference rule to all individuals in a set at once, rather than to each individual separately. This allows to apply the join order optimization, which uses database statistics to compute the data access path promising the least cost [1]. Second, the magic sets transformation [4] can be used to identify the subset of the database relevant to the query; this transformation was recently generalized to disjunctive programs [6].

$\mathcal{SHIQ}(\mathbf{D})$ is a complex logic, with modeling primitives such as number restrictions and concrete datatypes. This makes the reduction algorithm fairly complex, mainly because it is based on *basic superposition* [3,18]—a sophisticated calculus for theorem proving with equality. In this paper, we present the algorithm scaled down to the basic description logic $\mathcal{ALC}$. This makes the algorithm much simpler; in particular, it is based on the well-known ordered resolution calculus [2]. However, $\mathcal{ALC}$ exhibits features characteristic of most DLs, such as boolean concept constructors, existential and universal quantification, as well as general concept inclusion axioms, so this simplified algorithm succinctly demonstrates the important points of the general one.

Our results seem to integrate two fundamentally different logics. Therefore, in this paper we also answer common questions about the apparent mismatch in complexity between the two formalisms, or why a logic based on general first-order semantics and without unique name assumption can be embedded into a logic based on minimal-model semantics and unique name assumption.

2 Preliminaries

In this section we introduce the formalisms used in the paper.

2.1 Description Logics

Let N_R be a set of *roles*, and N_C a set of *concept names*. The set of $\mathcal{ALC}$ concepts is the minimal set satisfying the following conditions: (i) $\top$ and $\bot$ are $\mathcal{ALC}$ concepts; (ii) each $A \in N_C$ is an $\mathcal{ALC}$ concept; and (iii) $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, and $\exists R.C$ are $\mathcal{ALC}$ concepts, for C and D $\mathcal{ALC}$ concepts and R

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

Translating Concepts to FOL	
$\pi_y(\top, X) = \top$	$\pi_y(\bot, X) = \bot$
$\pi_y(A, X) = A(X)$	$\pi_y(\neg C, X) = \neg \pi_y(C, X)$
$\pi_y(C \sqcap D, X) = \pi_y(C, X) \wedge \pi_y(D, X)$	$\pi_y(C \sqcup D, X) = \pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X) = \forall y : R(X, y) \rightarrow \pi_x(C, y)$	$\pi_y(\exists R.C, X) = \exists y : R(X, y) \wedge \pi_x(C, y)$
Translating Axioms to FOL	
$\pi(C(a)) = \pi_y(C, a)$	$\pi(R(a, b)) = R(a, b)$
$\pi(C \sqsubseteq D) = \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$	$\pi(KB) = \bigwedge_{\alpha \in KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} \pi(\alpha)$
X is a meta variable and is substituted by the actual variable. π_x is obtained from π_y by simultaneously substituting in the definition $x_{(i)}$ for all $y_{(i)}$, respectively, and π_y for π_x.	

a role. Concept names are also called *atomic concepts*, and other concepts are also called *complex concept*. A *literal* concept is a possibly negated concept name. A TBox $KB_{\mathcal{T}}$ is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$. An ABox $KB_{\mathcal{A}}$ is a finite set of axioms $C(a)$ and $R(a, b)$, for C an $\mathcal{ALC}$ concept and R a role. An $\mathcal{ALC}$ knowledge base KB is a tuple $(KB_{\mathcal{T}}, KB_{\mathcal{A}})$.

The semantics of KB is given by translating it into first-order logic by the operator π from Table 1. The main inference problem is checking KB satisfiability—that is, determining if a first-order model of $\pi(KB)$ exists. An individual a is an *instance* of a concept C w.r.t. KB if $\pi(KB) \models \pi_y(C, a)$, which is the case if and only if $KB \cup \{\neg C(a)\}$ is unsatisfiable.

We measure the *size* of concepts by their length. An $\mathcal{ALC}$ concept is in *negation-normal form* (NNF) if negations occur only in front of concepts names. An ABox is *extensionally reduced* if all ABox axioms contain only literal concepts. If KB is not extensionally reduced, it can be easily transformed into an extensionally reduced knowledge base: for each axiom $C(a)$ where C is not a literal concept, one should introduce a new atomic concept A_C, add the axiom $A_C \sqsubseteq C$ to the TBox, and replace $C(a)$ with $A_C(a)$. Such a transformation is obviously polynomial, so without losing generality it is safe to assume that a knowledge base is extensionally reduced.

We adapt the notions of positions to DL formulae. A *position* p is a finite sequence of integers; the empty position is denoted with ϵ. If a position p_1 is a proper prefix of a position p_2, then p_1 is *above* p_2, and p_2 is *below* p_1. For a concept α, the subterm at a position p, written $\alpha|_p$, is defined as follows: $\alpha|_\epsilon = \alpha$; $(\neg D)|_{1p} = D|_p$; $(D_1 \circ D_2)|_{ip} = D_i|_p$ for $\circ \in \{\sqcap, \sqcup\}$ and $i \in \{1, 2\}$; and $\alpha|_1 = R$ and $\alpha|_{2p} = D|_p$ for $\alpha = \Diamond R.D$ and $\Diamond \in \{\exists, \forall\}$. A *replacement* of a subterm of α at position p with a term β is defined in the standard way and is denoted as $\alpha[\beta]_p$.

2.2 Disjunctive Datalog

A *datalog term* is a constant or a variable, and a *datalog atom* has the form $A(t_1, \ldots, t_n)$, where t_i are datalog terms. A *disjunctive datalog program* P is a finite set of rules of the form $A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$ where A_i and B_j are datalog atoms. 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 $A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m$. 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.

2.3 Ordered Resolution

We assume the reader to be familiar with standard definitions of first-order logic and clausal theorem proving. Ordered resolution [2] is one of the most widely used calculi for theorem proving in first-order logic. It is parameterized with an *admissible* ordering $\succ$ on literals (for details about the condition of admissibility please refer to [2]), and by a *selection function* S which assigns to each clause C a possibly empty subset of negative literals of C. With $\mathcal{R}$ we denote the calculus consisting of the following inference rules:

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

where (i) $\sigma = \mathsf{MGU}(A, B)$, (ii) $A\sigma$ is strictly 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 $D\sigma \vee \neg B\sigma$.

The clauses $C \vee A \vee B$ and $D \vee \neg B$ are the *main premises*, $C \vee A$ is the *side premise*, and $C\sigma \vee A\sigma$ and $C\sigma \vee D\sigma$ are the *conclusions*. Ordered resolution is sound and complete: if a set of clauses N is saturated up to redundancy by $\mathcal{R}$, then N is satisfiable if and only if it does not contain the empty clause.

3 The Main Difficulty in Reducing DLs to Datalog

For an $\mathcal{ALC}$ knowledge base KB, our goal is to derive a disjunctive datalog program $\mathsf{DD}(KB)$ such that $KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models \alpha$, for α of

the form $A(a)$ or $R(a, b)$. In other words, KB and $\mathsf{DD}(KB)$ should entail the same set of positive ground facts. We may thus use $\mathsf{DD}(KB)$ instead of KB for query answering, and in doing so we may apply all optimization techniques known from the field of deductive databases.

As shown by the definition of $\mathcal{ALC}$ and by Borgida [5], there is a close correspondence between description logics and first-order logic. Consider a knowledge base $KB = \{A \sqsubseteq \exists R.A,\ \exists R.\exists R.A \sqsubseteq B,\ A(a)\}$. A naïve attempt to reduce KB into disjunctive datalog is to compute a first-order formula $\pi(KB)$, skolemize it, translate it into conjunctive normal form, and rewrite the obtained set of clauses as rules, yielding the following logic program $\mathsf{LP}(KB)$:

$$R(x, f(x)) \leftarrow A(x) \tag{1}$$

$$A(f(x)) \leftarrow A(x) \tag{2}$$

$$B(x) \leftarrow R(x, y), R(y, z), A(z) \tag{3}$$

$$A(a) \tag{4}$$

Clearly, KB and $\mathsf{LP}(KB)$ entail the same set of ground facts. However, $\mathsf{LP}(KB)$ contains a function symbol in a recursive rule (2). This raises the issue of how to answer queries in $\mathsf{LP}(KB)$. Namely, well-known query evaluation techniques, such as bottom-up saturation, will not terminate on $\mathsf{LP}(KB)$: 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. Note that we need all previously derived facts to derive $B(a)$ from $\mathsf{LP}(KB)$, and that we do not know when all relevant ground facts have been derived.

To eliminate potential problems with termination, our goal is to derive a true disjunctive datalog program $\mathsf{DD}(KB)$—that is, a program without function symbols. For such a program, queries can be evaluated using any standard technique; furthermore, all known optimization strategies can be applied. Hence, the main problem that we address is how to eliminate function symbols from $\mathsf{LP}(KB)$.

To obtain the desired reduction, we start off with a slightly simpler task of deriving a program $\mathsf{DD}(KB)$ that is satisfiable if and only if KB is satisfiable. This we base on the following simple idea. Let us assume that unsatisfiability of KB can be demonstrated by a refutation in some sound and complete calculus $\mathcal{C}$. If it is possible to simulate inferences of $\mathcal{C}$ on KB using a datalog program $\mathsf{DD}(KB)$, a refutation in KB by $\mathcal{C}$ can be reduced to a refutation in $\mathsf{DD}(KB)$. Conversely, if $\mathsf{DD}(KB)$ is unsatisfiable, there is a refutation in $\mathsf{DD}(KB)$. If it is possible to simulate inferences in $\mathsf{DD}(KB)$ by the calculus $\mathcal{C}$ on KB, then a refutation in $\mathsf{DD}(KB)$ can be reduced to a refutation in KB.

To obtain a sound, complete, and terminating algorithm, we must select an appropriate calculus $\mathcal{C}$, capable of effectively deciding satisfiability of KB. Disjunctive datalog is strongly related to clausal first-order logic, so simulating inferences is easier if $\mathcal{C}$ is a clausal refutation calculus. Hence, to obtain a reduction algorithm, we first derive a decision procedure for $\mathcal{ALC}$ based on the ordered resolution calculus $\mathcal{R}$. In particular, in Section 4.1 we show how

to translate KB into an equisatisfiable set of clauses $\Xi(KB)$, and in Section 4.2 we show that exhaustive application of the inference rules of $\mathcal{R}$ on $\Xi(KB)$ eventually terminates. Since $\mathcal{R}$ is sound and complete, $\mathcal{R}$ decides satisfiability of $\Xi(KB)$, and therefore of KB as well.

Based on such a procedure, in Section 5 we derive the desired reduction of KB to a disjunctive datalog program. It turns out that, for $\mathcal{ALC}$, simulating inferences of $\mathcal{R}$ in disjunctive datalog is quite straightforward. After presenting several simple examples in Section 6, in Section 7 we discuss some interesting aspects of our algorithms.

4 Deciding Satisfiability of KB by Resolution

We now present an algorithm for deciding satisfiability of KB by ordered resolution. In the following sections, with Cls we denote the standard clausification operator: for a first-order formula φ, $\mathsf{Cls}(\varphi)$ is the set of clauses obtained by solemizing φ and translating it into conjunctive normal form.

4.1 Translating KB into Clauses

The first step in deciding satisfiability of an $\mathcal{ALC}$ knowledge base KB by the ordered resolution calculus $\mathcal{R}$ is to transform KB into an equisatisfiable set of clauses $\Xi(KB)$. A straightforward way to do so is to apply the operator Cls; however, such an algorithm has two important drawbacks. First, the size of $\mathsf{Cls}(\pi(KB))$ could be exponential in the size of $\pi(KB)$, due to nesting of $\sqcap$ and $\sqcup$ connectives. Second, since KB is an $\mathcal{ALC}$ knowledge base, the formula $\pi(KB)$ is of a particular syntactic structure, which we exploit in the decision procedure. Therefore, we apply the *structural transformation*, introduced in [19]. Next, we present an alternative, but equivalent definition.

Definition 4.1 Let C be an $\mathcal{ALC}$ concept in negation-normal form. A position $p \neq \epsilon$ in C is *eligible for replacement* if $C|_p$ is of the form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, where $L_{(i)}$ are all literal concepts. The definitorial form of C, written $\mathsf{Def}(C)$, is defined recursively as follows:

$$\mathsf{Def}(C) = \begin{cases} \{C\} & \text{if } C \text{ is a literal concept} \\ \{\neg Q \sqcup C|_p\} \cup \mathsf{Def}(C[Q]_p) & \text{if } p \text{ is eligible for replacement in } C \end{cases}$$

For example, $\mathsf{Def}(\exists R.(\forall S.\neg A)) = \{\exists R.Q,\ \neg Q \sqcup \forall S.\neg A\}$. Note that the concept $\neg Q \sqcup \forall S.\neg A$ can be interpreted as $Q \sqsubseteq \forall S.\neg A$, which defines Q as a new name for $\forall S.\neg A$. Furthermore, as shown by the following lemma, this transformation does not affect satisfiability.

Lemma 4.2 *For a concept C in negation-normal form, the axiom $\top \sqsubseteq C$ is satisfiable if and only if the set of axioms $\{\top \sqsubseteq D_i \mid D_i \in \mathsf{Def}(C)\}$ is satisfiable.*

Table 2
Types of $\mathcal{ALC}$-clauses

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

Proof. We apply induction on the number of recursive invocations of Def. The induction base is trivial, so we consider the induction step.

($\Rightarrow$) Let I be a model satisfying $\top \sqsubseteq C$, and let I' be an extension of I such that $Q^{I'} = (C|_p)^I$. Obviously, I' satisfies $\top \sqsubseteq C[Q]_p$ and $\top \sqsubseteq \neg Q \sqcup C|_p$.

($\Leftarrow$) Observe that in each model I of $\top \sqsubseteq \neg Q \sqcup C|_p$, we have $Q^I \subseteq (C|_p)^I$. By induction on the structure of $D = C[Q]_p$, it is easy to show that, for each position q in D, $(D|_q)^I \subseteq (C|_q)^I$. Since $\triangle^I \subseteq (D|_q)^I$ for each q, I is a model of $\top \sqsubseteq C$. □

The set of clauses $\Xi(KB)$, encoding an $\mathcal{ALC}$ knowledge base KB in first-order logic, is defined as follows:

Definition 4.3 The operator Cls is extended to $\mathcal{ALC}$ concepts as follows:

$$\mathsf{Cls}(C) = \bigcup_{D \in \mathsf{Def}(\mathsf{NNF}(C))} \mathsf{Cls}(\forall x : \pi_y(D, x))$$

For an extensionally reduced $\mathcal{ALC}$ knowledge base KB, $\Xi(KB)$ is the smallest set of clauses such that (*i*) $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$ for each ABox axiom α in KB; and (*ii*) $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$ for each TBox axiom $C \sqsubseteq D$ in KB. If KB is not extensionally reduced, then $\Xi(KB) = \Xi(KB')$, where KB' is an extensionally reduced knowledge base obtained from KB as shown in Section 2.1.

We define *$\mathcal{ALC}$-clauses* to be clauses of the form from Table 2. For a term t, $\mathbf{P}(t)$ denotes a possibly empty disjunction $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$, and $\mathbf{P}(\mathbf{f}(x))$ denotes a possibly empty disjunction $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$. Note that this allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals.

We now show that clausification does not affect the semantics of a knowledge base, and that it produces only $\mathcal{ALC}$ clauses:

Lemma 4.4 *Let KB be an $\mathcal{ALC}$ knowledge base. Then, (*i*) KB is satisfiable if and only if $\Xi(KB)$ is satisfiable; (*ii*) $\Xi(KB)$ can be computed in time polynomial in $|KB|$; and (*iii*) each clause in $\Xi(KB)$ is an $\mathcal{ALC}$ clause.*

Proof. Equisatisfiability of KB and $\Xi(KB)$ is an easy consequence of Lemma 4.2. Furthermore, the number of recursive invocations of Def, and the number

of new concepts Q are linear in the number of subconcepts of C. Hence, $|\mathsf{Def}(C)|$ is linear in $|C|$, so $|\Xi(KB)|$ is polynomial in $|KB|$. Finally, observe that $\mathsf{Def}(C)$ can contain only concepts of the form D or $\neg Q \sqcup D$, where D is of the form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, and all $L_{(i)}$ are literal concepts. Hence, for $D = \exists R.L$, $\mathsf{Cls}(D)$ contains clauses of type 1 and 2; for $D = \forall R.L$ a clause of type 3; and for $D = \bigsqcup L_i$ or $D = \bigsqcap L_i$ clauses of type 2. Clauses of types 4 and 5 are obtained by clausifying ABox axioms. □

4.2 Deciding Satisfiability of $\Xi(KB)$ by $\mathcal{R}$

Since $\mathcal{R}$ is a sound and complete calculus, we can use it to check satisfiability of $\Xi(KB)$. To obtain a decision procedure, we just need to ensure that each saturation of $\Xi(KB)$ by $\mathcal{R}$ terminates. The basic principle to achieve this has been outlined by Joyner [16]: we simply ensure that the number of clauses that can be derived in a saturation is bounded.

There are two main reasons why a calculus might derive an infinite number of clauses in a saturation. First, we might keep deriving clauses with deeper and deeper terms. Consider $N_1 = \{C(a), \neg C(x) \vee C(f(x))\}$. If we select $\neg C(x)$ in the second clause and apply resolution, we shall derive $C(f(a))$, $C(f(f(a)))$, and so forth. Second, we might keep deriving clauses with more and more variables. Consider $N_2 = \{\neg R(x,y) \vee \neg R(y,z) \vee R(x,z), C(x) \vee R(x,y) \vee D(y), E(x) \vee R(x,y) \vee F(y)\}$. If we select $\neg R(x,y)$ and $\neg R(y,z)$ in the first clause, by resolution we obtain $C(x) \vee D(y) \vee E(y) \vee F(z) \vee R(x,z)$, which contains more variables than the side premises involved in the inference; further inferences will additionally increase the number of variables.

By choosing these parameters appropriately, it is possible to restrict the resolution inferences, allowing us to establish a bound on the term depth and on the number of variables. Consider again the set of clauses N_1: instead of selecting $\neg C(x)$, if we ensure that $C(f(x)) \succ \neg C(x)$, then the second clause can participate in an inference only on literal $C(f(x))$. Furthermore, $C(f(x))$ and $C(a)$ do not unify, so no inference of $\mathcal{R}$ is applicable to N_1. Hence, N_1 is saturated, and, since it does not contain the empty clause, it is satisfiable. In the following definition we choose the parameters for $\mathcal{R}$ that achieve such an effect on $\mathcal{ALC}$-clauses.

Definition 4.5 Let $\mathcal{R}_{DL}$ denote the calculus $\mathcal{R}$ parameterized as follows:

- In the literal ordering, $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$, for all function symbols f, and predicates R, C, and D.
- The selection function selects in each clause every negative binary literal.

We now prove the following central lemma:

Lemma 4.6 *Each $\mathcal{R}_{DL}$ inference, when applied to $\mathcal{ALC}$-clauses, produces an $\mathcal{ALC}$-clause.*

Table 3
Possible Inferences by $\mathcal{R}_{DL}$ on $\mathcal{ALC}$-clauses

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(x) \vee \mathbf{P_3}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(g(x))} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \neg A(x) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(g(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee R(x, f(x)) \quad \mathbf{P_2}(x) \vee \neg R(x, y) \vee \mathbf{P_3}(y)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x) \vee \mathbf{P_3}(f(x))} \textbf{ (1+3=2)}$$

$$\frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(b)} \textbf{ (4+2=4)} \qquad \frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(b) \vee \mathbf{P_2}(\mathbf{c})}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(\mathbf{c})} \textbf{ (4+4=4)}$$

$$\frac{R(a, b) \quad \mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)}{\mathbf{P_1}(a) \vee \mathbf{P_2}(b)} \textbf{ (5+3=4)} \qquad \frac{R(a, b) \quad \neg R(a, b)}{\Box} \textbf{ (5+5=2)}$$

Proof. The lemma can easily be proved by considering all possible $\mathcal{R}_{DL}$ inferences on all types of $\mathcal{ALC}$-clauses, which are summarized in Table 3. For the sake of brevity, we omit inferences in which literals participating in inferences are complemented. The notation $(n + m = k)$ next to each inference specifies that the inference premises are of types n and m, and the conclusion is of type k. Observe that, 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 $f(x)$. $\Box$

We now show that, for a finite knowledge base KB, the number of $\mathcal{ALC}$-clauses is finite. In fact, this bound can be used to estimate the complexity of the algorithm.

Lemma 4.7 *For an $\mathcal{ALC}$ knowledge base KB, the longest $\mathcal{ALC}$-clause over the signature of $\Xi(KB)$ is polynomial in $|KB|$, and the number of such clauses different up to variable renaming is exponential in $|KB|$.*

Proof. Let c be the number of unary predicates, r the number of binary predicates, f the number of unary function symbols, and i the number of constants in the signature of $\Xi(KB)$. Then, c is linear in $|KB|$, since each concept introduced in $\mathsf{Def}(C)$ corresponds to one nonliteral subconcept of C.

Similarly, f is linear in $|KB|$, since each function symbol is introduced by skolemizing one concept $\exists R.C$. Finally, i is trivially linear in $|KB|$.

Consider now the the maximal $\mathcal{ALC}$-clause C_m. It may contain at most $t = 2+i+f$ terms: a term can be the variable x, the variable y, an individual, or of the form $f(x)$. Hence, C_m contains at most $\ell = 2ct + 2rt^2$ literals (the factor 2 takes into account that each literal can be either positive or negative), which is polynomial in $|KB|$. Each $\mathcal{ALC}$-clause is a subset of C_m, so there are 2^ℓ such clauses; that is, the number of clauses is exponential in $|KB|$. □

We now state the main result of this subsection:

Theorem 4.8 *For an $\mathcal{ALC}$ knowledge base KB, saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$ decides satisfiability of KB and runs in time exponential in $|KB|$.*

Proof. By Lemma 4.7, the number of clauses derivable by $\mathcal{R}_{DL}$ from $\Xi(KB)$ is exponential in $|KB|$. Each inference step 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 $\Xi(KB)$, and by Lemma 4.4 of KB as well, in time exponential in $|KB|$. □

5 Translating $\mathcal{ALC}$ to Disjunctive Datalog

Given a decision procedure for checking satisfiability of an $\mathcal{ALC}$ knowledge base KB, it is now easy to obtain the desired reduction to disjunctive datalog. From Table 3 we see that (*i*) a ground clause cannot participate in an inference with a nonground clause containing function symbols, and (*ii*) as soon as one premise in an inference by $\mathcal{R}_{DL}$ is ground, the conclusion is ground as well. Hence, we may perform all inferences among nonground clauses first, after which we may delete all nonground clauses containing function symbols and rewrite clauses as rules. A minor problem arises if thus obtained rules contain unsafe variables; we deal with them by adding a special literal with a predicate HU, which explicitly enumerates the Herbrand universe.

Definition 5.1 Let KB be an $\mathcal{ALC}$ knowledge base. Then, $\Gamma(KB_\mathcal{T})$ is the set of clauses obtained by saturating $\Xi(KB_\mathcal{T})$ by $\mathcal{R}_{DL}$, and then deleting all clauses containing function symbols. Furthermore, the operator λ maps clauses to clauses as follows:

$$\lambda(C) = C \cup \{\neg HU(x) \mid \text{ for each unsafe variable } x \text{ in } C\ \}$$

For a set of clauses N, $\lambda(N)$ is the set of clauses obtained by applying λ to each element of N. The function-free version of KB is defined as follows:

$$\mathsf{FF}(KB) = \lambda(\Gamma(KB_\mathcal{T})) \cup \Xi(KB_\mathcal{A}) \cup \{HU(a) \mid \text{ for each individual } a\ \}$$

Finally, a disjunctive datalog program $\mathsf{DD}(KB)$ is the set of rules obtained by moving in each clause from $\mathsf{FF}(KB)$ all positive literals into the rule head, and all negative literals into the rule body.

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

Theorem 5.2 *For an $\mathcal{ALC}$ knowledge base KB, the following claims hold:*

(i) *KB is unsatisfiable if and only if $\mathsf{DD}(KB)$ is unsatisfiable.*

(ii) *$KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, where α is of the form $A(a)$ or $R(a, b)$, and A is an atomic concept.*

(iii) *$KB \models C(a)$ for a nonliteral concept C if and only if, for Q a new atomic concept, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\}) \models_c Q(a)$.*

(iv) *The number of literals in each rule in $\mathsf{DD}(KB)$ is at most polynomial, the number of rules in $\mathsf{DD}(KB)$ is at most exponential, and $\mathsf{DD}(KB)$ can be computed in time exponential in $|KB|$.*

Proof. (Claim i) Table 3 shows that each inference with at least one ground premise always produces a ground conclusion. Hence, in saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$, all inferences among nonground clauses can be performed first. Furthermore, Table 3 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 $\Xi(KB)$, one may delete all clauses containing terms of the form $f(x)$.

By Definition 4.3, $\Xi(KB_{\mathcal{T}})$ is exactly the set of nonground clauses of $\Xi(KB)$, so $\Gamma(KB_{\mathcal{T}})$ is exactly the set of clauses obtained by saturating the nonground part of $\Xi(KB)$, and deleting the clauses containing function symbols. Furthermore, it is easy to see that $\Gamma = \Gamma(KB_{\mathcal{T}}) \cup \Xi(KB_{\mathcal{A}})$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable. Namely, both Γ and $\mathsf{FF}(KB)$ are function-free sets of clauses, whose sets of ground instances differ only on clauses containing unsafe variables. However, for a clause $C \in \Gamma$ containing an unsafe variable x and a ground substitution $\tau = \{x \mapsto a\}$, $\mathsf{FF}(KB)$ contains clauses $C \vee \neg HU(x)$ and $HU(a)$, which together imply $C\tau$. Hence, the ground instances of $\mathsf{FF}(KB)$ imply all ground instances of Γ, so, if $\mathsf{FF}(KB)$ is satisfiable, Γ is satisfiable as well. Furthermore, since HU is a new predicate not occurring in Γ, each model of Γ can easily be extended to a model of $\mathsf{FF}(KB)$. Hence, we conclude that Γ and $\mathsf{FF}(KB)$ are equisatisfiable.

Finally, $\mathsf{DD}(KB)$ is a positive datalog program whose rules are syntactic variants of the clauses from $\mathsf{FF}(KB)$. Hence, $\mathsf{DD}(KB)$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable.

(Claim ii) Observe that $KB \models \alpha$ if and only if $KB \cup \{\neg\alpha\}$ is unsatisfiable. The latter is the case if and only if $\mathsf{DD}(KB \cup \{\leftarrow \alpha\}) = \mathsf{DD}(KB) \cup \{\leftarrow \alpha\}$ is unsatisfiable, which is the case if and only if $\mathsf{DD}(KB) \models_c \alpha$.

(Claim iii) Follows in the same manner as Claim ii.

(Claim iv) Follows immediately from Lemma 4.7. □

6 Examples

We now present several rather simple examples, which point out important properties of the reduction algorithm.

Readers familiar with more common DL reasoning algorithms might wonder how are role successors represented in a datalog program. Consider the knowledge base $KB_1 = \{C \sqsubseteq \exists R.D\}$; the corresponding set of clauses is $\Xi(KB_1) = \{\neg C(x) \vee R(x, f(x)), \neg C(x) \vee D(f(x))\}$. Now $\Xi(KB_1)$ is already saturated by $\mathcal{R}_{DL}$, so after removing all clauses containing function symbols, we get $\mathsf{DD}(KB_1) = \emptyset$. This may seem quite confusing: KB_1 implies the existence of at least one R-successor for each member of C, whereas $\mathsf{DD}(KB_1)$ does not reflect that. However, Theorem 5.2 is not invalidated. Namely, the individuals introduced by the existential quantifier in KB_1 are unnamed, so they cannot be used in queries. Hence, for an arbitrary extensionally reduced ABox $KB_{\mathcal{A}}$, $KB_1 \cup KB_{\mathcal{A}}$ does not imply any new facts of the form $A(a)$ or $R(a, b)$. Also, note that the models of $\mathsf{DD}(KB_1)$ are completely unrelated to the models of KB_1. Intuitively, this is so because the reduction algorithm is based on a proof-theoretic correspondence between refutations in $\Xi(KB)$ and $\mathsf{DD}(KB)$. The models of KB and $\mathsf{DD}(KB)$ coincide only on positive ground facts, whereas for unnamed individuals they are completely unrelated.

In order to be able to draw additional consequences from KB_1, we must extend it with additional axioms. For example, $KB_2 = KB_1 \cup \{D \sqsubseteq \bot\}$ yields $\Xi(KB_2) = \Xi(KB_1) \cup \{\neg D(x)\}$. Saturation of $\Xi(KB_2)$ produces one additional clause $\neg C(x)$, so we get $\mathsf{DD}(KB_2) = \{\leftarrow C(x), \leftarrow D(x)\}$. This shows that the reduction is not modular: for arbitrary knowledge bases KB_1 and KB_2, $\mathsf{DD}(KB_1) \cup \mathsf{DD}(KB_2)$ is not necessarily equal to $\mathsf{DD}(KB_1 \cup KB_2)$.

The key step in the reduction is the saturation of $\Xi(KB_{\mathcal{T}})$ by $\mathcal{R}_{DL}$. It computes all relevant nonground consequences of $\Xi(KB_{\mathcal{T}})$, ensuring that subsequent removal of clauses containing function symbols does not change the set of ground consequences. One may think of $\mathcal{R}_{DL}$ as producing shortcut clauses, which derive facts about objects without explicitly expanding the successors of each object. Consider $KB_3 = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq C,\ \exists R.C \sqsubseteq D\}$, producing the following clauses in $\Xi(KB_3)$:

$$\neg A(x) \vee R(x, f(x)) \tag{5}$$

$$\neg A(x) \vee B(f(x)) \tag{6}$$

$$\neg B(x) \vee C(x) \tag{7}$$

$$D(x) \vee \neg R(x, y) \vee \neg C(y) \tag{8}$$

Assuming a literal ordering such that $D(x) \succ C(x) \succ B(x) \succ A(x)$, by saturating $\Xi(KB_3)$ we obtain the following new clauses (the notation $R(xx; yy)$ means that a clause is derived by resolving clauses xx and yy):

$$\neg A(x) \vee D(x) \vee \neg C(f(x)) \quad \text{R(5;8)} \tag{9}$$
$$\neg A(x) \vee D(x) \vee \neg B(f(x)) \quad \text{R(9;7)} \tag{10}$$
$$\neg A(x) \vee D(x) \quad \text{R(10;6)} \tag{11}$$

Eliminating clauses with function symbols yields $DD(KB_3)$ as follows:

$$C(x) \leftarrow B(x) \tag{12}$$
$$D(x) \leftarrow R(x, y), C(y) \tag{13}$$
$$D(x) \leftarrow A(x) \tag{14}$$

It is instructive to consider the role of each rule in $\mathsf{DD}(KB_3)$. Whereas the axiom $B \sqsubseteq C$ in KB_3 is applicable to all individuals in a model, the rule (12) is applicable only to named individuals. The relationship between $\exists R.C \sqsubseteq D$ and (13) is analogous. However, (12) and (13) derive consequences only about named individuals, so $\mathsf{DD}(KB_3)$ contains the rule (14), which is produced in the saturation of $\Xi(KB_{\mathcal{T}})$ by $\mathcal{R}_{DL}$. This rule can be thought of as a shortcut: instead of introducing for each x in A an R-successor y in B, propagating y to C, and then concluding that x is in D, (14) derives that all members of A are members of D in one step, thus ensuring that $\mathsf{DD}(KB_3)$ and KB_3 entail the same set of ground facts.

Finally, we take KB_4 to be the knowledge base introduced in Section 3. The concept $\exists R.\exists R.A$ contains a nonatomic subconcept $\exists R.A$, so we apply structural transformation, and replace $\exists R.A$ with a new atomic concept Q. This yields the following set of clauses $\Xi(KB_4)$:

$$\neg A(x) \vee R(x, f(x)) \tag{15}$$
$$\neg A(x) \vee A(f(x)) \tag{16}$$
$$Q(x) \vee \neg R(x, y) \vee \neg A(y) \tag{17}$$
$$B(x) \vee \neg R(x, y) \vee \neg Q(y) \tag{18}$$

Assuming a literal ordering where $Q(x) \succ B(x) \succ A(x)$, a saturation of $\Xi(KB_4)$ by $\mathcal{R}_{DL}$ produces the following new clauses:

$$\neg A(x) \vee Q(x) \vee \neg A(f(x)) \quad \text{R(15;17)} \tag{19}$$
$$\neg A(x) \vee Q(x) \quad \text{R(19;16)} \tag{20}$$
$$\neg A(x) \vee B(x) \vee \neg Q(f(x)) \quad \text{R(15;18)} \tag{21}$$
$$\neg A(x) \vee B(x) \vee \neg A(f(x)) \quad \text{R(21;20)} \tag{22}$$
$$\neg A(x) \vee B(x) \quad \text{R(22;16)} \tag{23}$$

Eliminating clauses with function symbols yields $\mathsf{DD}(KB)$ as follows:

$$Q(x) \leftarrow R(x,y), A(y) \tag{24}$$
$$B(x) \leftarrow R(x,y), Q(y) \tag{25}$$
$$Q(x) \leftarrow A(x) \tag{26}$$
$$B(x) \leftarrow A(x) \tag{27}$$

The intuitive meaning behind these rules is somewhat obscured because of the predicate Q. However, we are not interested in ground consequences related to Q, and, since Q is a new predicate, it cannot occur in any ABox one might consider in conjunction with KB_4. Hence, it is possible to apply rule unfolding to Q, yielding the following datalog program:

$$B(x) \leftarrow A(x) \tag{28}$$
$$B(x) \leftarrow R(x,y), A(y) \tag{29}$$
$$B(x) \leftarrow R(x,y), R(y,z), A(z) \tag{30}$$

Now the intuition behind these rules can be explained as follows. The rule (28) takes into account that each named individual, which is in A, has a chain of at least two unnamed R-successors. The rule (29) takes into account that a named individual may be explicitly linked through R to another individual in A, which then has at least one unnamed R-successor. Finally, the rule (30) takes into account that a named individual may be explicitly linked through an R-chain of length two to an individual which is in A. Only R-chains of length two are considered, because KB_4 contains the concept $\exists R.\exists R.A$, which effectively checks for an R-chain of length two.

7 Discussion

In this subsection we discuss some aspects of our algorithms that may seem surprising at the first glance.

7.1 Independence of the Reduction from the Query

Theorem 5.2 shows that the program $\mathsf{DD}(KB)$ is independent of the query, as long as the query is a positive atomic concept or a role. Hence, $\mathsf{DD}(KB)$ can be computed once, and can be used to answer any number of atomic queries.

On the contrary, if the query involves a nonatomic concept C (even if C is a negated atomic concept), then query answering must 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, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\})$ depends on the query concept C. Intuitively, a complex concept C used in the query introduces a kind of terminological knowledge, so the reduction is not independent from complex queries.

7.2 Complexity

The complexity of cautious query answering in a nonground disjunctive datalog program P is in co-NEXPTIME$^{\text{NP}}$[7]. Since the number of rules in $|P|$ can be exponential in $|KB|$, one might think that reducing DLs to disjunctive datalog significantly increases the complexity of reasoning. However, we show that this is not the case. Let P^g_{DL} be a program obtained by replacing in $\mathsf{DD}(KB)$ all variables with individuals in all possible ways. The number of variables in a rule from $\mathsf{DD}(KB)$ is at most two, so $|P^g_{DL}|$ is exponential in $|KB|$. Furthermore, the arity of predicates in $\mathsf{DD}(KB)$ is at most two, so the number of ground atoms in P^g_{DL} is polynomial in $|KB|$. Hence, an interpretation I_{DL} for P^g_{DL} can be guessed in time which is polynomial in $|KB|$ by choosing a subset of the ground atoms; furthermore, all such interpretations can be examined in time which is exponential in $|KB|$. Checking if I_{DL} is a model of P^g_{DL} can be performed in time which is exponential in $|KB|$. These two steps combined give us the EXPTIME complexity for satisfiability checking, and also for unsatisfiability checking and query answering. To summarize, even though $|\mathsf{DD}(KB)|$ is exponential in $|KB|$, query answering can be performed in exponential time because (*i*) the length of rules in $\mathsf{DD}(KB)$ is polynomial in $|KB|$, (*ii*) the arity of predicates is bounded, and (*iii*) checking minimality of an interpretation is not necessary for positive queries.

7.3 Minimal vs. Arbitrary Models

Disjunctive datalog programs are usually interpreted under *minimal model* semantics: $P \models_c \alpha$ means that α is true in all minimal models of a program P, where minimality is defined w.r.t. set inclusion. However, description logics assume the standard first-order semantics: $KB \models \alpha$ means that α is true in all models of KB. By minimal models, disjunctive datalog implements a kind of *closed-world* semantics; on the contrary, description logics implement *open-world* semantics. Because these two semantics are quite different, our result may seem surprising.

For P a positive datalog program and α a positive ground atom, $P \models \alpha$ 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.

For α a negative ground atom, the difference between minimal model semantics and first-order semantics is relevant. For 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 $P \models_c \alpha$.

Subsumption also depends on the type of chosen semantics. For example, let $\alpha = \forall x : [C(x) \rightarrow D(x)]$ and $P = \{C(a), D(a)\}$. Then, $P \not\models \alpha$: just

consider a first-order 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 [20], 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 answering positive queries in positive datalog programs; however, it is relevant for queries which involve negation or for concept subsumption. Theorem 5.2 reflects this: it says that all positive ground facts entailed by KB are contained in each minimal model of $\mathsf{DD}(KB)$ and vice versa, but does not make any other stronger statements.

It would therefore be incorrect to check whether $KB \models \neg A(a)$ by checking if $\mathsf{DD}(KB) \models_c \neg A(a)$. To apply our algorithm, we must reduce the problem to entailment of positive ground facts: for a new concept $NotA$, $KB \models \neg A(a)$ if and only if $\mathsf{DD}(KB \cup \{\neg A \sqsubseteq NotA\}) \models_c NotA(a)$.

Similarly, it would be incorrect to check whether $KB \models C \sqsubseteq D$ by checking whether $\mathsf{DD}(KB) \models_c \forall x : [C(x) \rightarrow D(x)]$. We must again reduce the problem to entailment of positive ground facts. If C and D are atomic concepts, by well-known transformations we may show that $KB \models C \sqsubseteq D$ if and only if $\mathsf{DD}(KB) \cup \{C(\iota)\} \models_c D(\iota)$, where ι is a new individual not occurring in KB.

7.4 Descriptive vs. Minimal-Model Semantics

Our reduction preserves the so-called descriptive semantics. Namely, Nebel showed that knowledge bases containing terminological cycles are not definitorial [17]: for a fixed partial interpretation of atomic concepts, several interpretations of nonatomic concepts may exist. In such a case, it might be reasonable to designate a particular interpretation as the intended one, with least and greatest fixpoint models being the obvious candidates. However, Nebel argues that it is not clear which interpretation best matches the intuition, because choosing either of the fixpoint models has its drawbacks. Consequently, most description logic systems implement the descriptive semantics, which coincides with the one we presented in Section 2.

By Theorem 5.2, our decision procedure implements exactly the descriptive semantics. Namely, $\mathsf{DD}(KB)$ entails exactly those ground facts which are derivable using the resolution decision procedure, and the latter implements the descriptive semantics.

7.5 Unique Name Assumption

The semantics of $\mathcal{ALC}$ and $\mathcal{SHIQ}$ does not require *unique name assumption* (UNA)—that is, that different symbols are interpreted as different objects; however, the semantics of disjunctive datalog does require it. Therefore, it

may seem surprising that a logic without UNA can be embedded into a logic which strictly requires it.

To understand why our algorithms are correct, note that UNA can be used to derive new consequences only if the theory employs equality. $\mathcal{ALC}$ provides neither number restrictions, nor explicit individual equality statements, so it actually does not require equality reasoning. Given a knowledge base KB, to enforce UNA we may append an axiom $a_i \not\approx a_j$ for each pair of different individuals a_i and a_j. However, since KB does not contain the equality predicate, these inequality axioms do not participate in any inference with $\mathcal{ALC}$-clauses, so they are not needed in the first place. A model-theoretic explanation is given by the following claim, which holds for any logic without equality: for each model I, in which distinct constants are interpreted by the same objects, there is a model I' in which all constants are interpreted by distinct objects. To summarize, in the case of $\mathcal{ALC}$, we simply do not care whether either KB or $\mathsf{DD}(KB)$ employs UNA, as this does not change the entailed set of facts.

The situation changes slightly for logics such as $\mathcal{SHIQ}$, which do require equality reasoning. Consider $KB = \{\top \sqsubseteq \leq 1\, R,\ R(a,b),\ R(a,c)\}$. Without UNA, KB is satisfiable, and $KB \models b \approx c$. Namely, the first axiom requires R to be functional, so b and c must be interpreted as the same object. On the contrary, KB is unsatisfiable with UNA.

A disjunctive datalog program $\mathsf{DD}(KB)$, which corresponds to KB, contains the rules (31)–(33). Note that $\mathsf{DD}(KB)$ is now a disjunctive datalog program with equality, which is quite different from the type of equality usually considered in disjunctive datalog.

$$y_1 \approx y_2 \leftarrow R(x, y_1), R(x, y_2) \tag{31}$$

$$R(a, b) \tag{32}$$

$$R(a, c) \tag{33}$$

In [7] the authors consider disjunctive datalog in which negated equality atoms can occur in rule bodies. Such programs are interpreted under UNA, so an inequality atom $\neg(x \approx y)$ actually checks whether x and y are bound to syntactically different individuals.

In our case, however, we allow equality in the rule heads as well. Thus, we can actually *derive* two individuals to be equal. Such inferences are not directly supported in disjunctive datalog; however, they can be simulated using the well-known encoding from [8], in which the $\approx$ is treated as an ordinary predicate, with required properties axiomatized explicitly. A disjunctive datalog program P with equality is thus transformed into a disjunctive datalog program $P_{\approx}$ without equality, by stating that $\approx$ is reflexive, symmetric, and transitive, and by appending the following replacement rule, instantiated for each distinct predicate R and position i in it:

$$R(x_1, \ldots, y_i, \ldots, x_n) \leftarrow R(x_1, \ldots, x_i, \ldots, x_n), x_i \approx y_i$$

Since $P_{\approx}$ is now a disjunctive datalog program without equality, we may interpret it either with or without UNA, just as we discussed previously. In each Herbrand model $I_{\approx}$ of $P_{\approx}$, the rules in $P_{\approx} \setminus P$ ensure that $\approx$ is interpreted as a congruence relation, so we can always transform $I_{\approx}$ to an interpretation I by replacing each individual a with the equivalence class of $\approx$ to which a belongs. It is well known that thus obtained I is a model of P.

7.6 The Size of $\mathsf{DD}(KB)$

By Theorem 5.2, the size of the rules of $|\mathsf{DD}(KB)|$ can be exponential in $|KB|$, which may seem discouraging in practice. We address this issue in two ways.

On the theoretical side, in [15] we showed that this blowup is only in the size of the TBox of KB. Hence, a sufficiently large ABox can actually dominate the size of the rules, which leads to novel *data complexity* results: in general, checking satisfiability of KB is NP-complete in $|KB_{\mathcal{A}}|$; furthermore, we identified a new fragment of $\mathcal{SHIQ}$ for which satisfiability checking is even polynomial in $|KB_{\mathcal{A}}|$.

On the practical side, in [13] we presented an important optimization which allows removing many rules from $|\mathsf{DD}(KB)|$ without jeopardizing completeness. Intuitively, the saturation of $\Xi(KB)$ introduces clauses which are entailed by other clauses. Such clauses are needed to derive all relevant clauses without function symbols; however, once the saturation has finished, they may be deleted. We have seen that this drastically reduces the number of rules in $\mathsf{DD}(KB)$ in practice.

8 Conclusion

In this paper, we present a version of our algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program [13,12,14], scaled down to the basic description logic $\mathcal{ALC}$. This simplified algorithm concisely conveys our basic idea, and may help to understand the general algorithms more easily. The reduction technique allows to perform DL reasoning using techniques developed for deductive databses, such as join order optimization or the magic sets transformation. The latter has been shown to dramatically improve the evaluation of disjunctive datalog programs, as it reduces the number of models of the disjunctive program.

We have implemented our algorithm in a new DL reasoner KAON2 [1], and the results of a performance evaluation are available on the project Web site. For knowledge bases with a relatively small TBox, but a large ABox, we have observed performance improvements over state-of-the-art tableau reasoners of one or more orders of magnitude. For TBox reasoning, our system is not able to solve all problems that tableau provers can; however, it is still able to classify certain nontrivial ontologies.

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

For our future work, we see two main challenges. On the practical side, the performance of the system might be further improved by applying heuristics based on data statistics. On the theoretical side, we shall investigate possibilities of extending DLs with some kind of nonmonotonic reasoning.

Acknowledgment

This work has been carried out in cooperation with Ullrich Hustadt and Ulrike Sattler.

References

[1] Abiteboul, S., R. Hull and V. Vianu, "Foundations of Databases," Addison Wesley, 1995.

[2] Bachmair, L. and H. Ganzinger, *Resolution Theorem Proving*, , **I**, Elsevier Science, 2001 pp. 19–99.

[3] Bachmair, L., H. Ganzinger, C. Lynch and W. Snyder, *Basic Paramodulation*, Information and Computation **121** (1995), pp. 172–192.

[4] Beeri, C. and R. Ramakrishnan, *On the power of magic*, in: *Proc. PODS '87* (1987), pp. 269–283.

[5] Borgida, A., *On the Relative Expressiveness of Description Logics and Predicate Logics*, Artificial Intelligence **82** (1996), pp. 353–367.

[6] Cumbo, C., W. Faber, G. Greco and N. Leone, *Enhancing the Magic-Set Method for Disjunctive Datalog Programs*, in: B. Demoen and V. Lifschitz, editors, *Proc. ICLP 2004*, LNCS **3132** (2004), pp. 371–385.

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

[8] Fitting, M., "First-Order Logic and Automated Theorem Proving, 2nd Edition," Texts in Computer Science, Springer, 1996.

[9] Haarslev, V. and R. Möller, *Optimization Strategies for Instance Retrieval*, in: I. Horrocks, S. Tessaris and J. Z. Pan, editors, *Proc. DL 2002*, CEUR Workshop Proceedings **53**, Toulouse, France, 2002.

[10] Horrocks, I., "Optimising Tableaux Decision Procedures for Description Logics," Ph.D. thesis, University of Manchester, UK (1997).

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

[12] Hustadt, U., B. Motik and U. Sattler, *Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution*, in: *Proc. ECAI 2004* (2004), pp. 353–357.

[13] Hustadt, U., B. Motik and U. Sattler, *Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs*, in: *Proc. KR 2004* (2004), pp. 152–162.

[14] Hustadt, U., B. Motik and U. Sattler, *A Decomposition Rule for Decision Procedures by Resolution-based Calculi*, in: *Proc. LPAR 2004*, LNAI **3452** (2005), pp. 21–35.

[15] Hustadt, U., B. Motik and U. Sattler, *Data Complexity of Reasoning in Very Expressive Description Logics*, in: *Proc. IJCAI 2005* (2005), pp. 466–471.

[16] Jr., W. H. J., *Resolution Strategies as Decision Procedures*, Journal of the ACM **23** (1976), pp. 398–417.

[17] Nebel, B., *Terminological Cycles: Semantics and Computational Properties*, in: J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991 pp. 331–361.

[18] Nieuwenhuis, R. and A. Rubio, *Theorem Proving with Ordering and Equality Constrained Clauses*, Journal of Symbolic Computation **19** (1995), pp. 312–351.

[19] Plaisted, D. A. and S. Greenbaum, *A Structure-Preserving Clause Form Translation*, Journal of Symbolic Logic and Computation **2** (1986), pp. 293–304.

[20] Shmueli, O., *Equivalence of DATALOG Queries is Undecidable*, Journal of Logic Programming **15** (1993), pp. 231–241.