Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs

Technical Report FZI 1-8-11/03, Revised Version

Ullrich Hustadt[1], Boris Motik[2], and Ulrike Sattler[3]

[1] Department of Computer Science, University of Liverpool
Liverpool, UK
U.Hustadt@csc.liv.ac.uk
[2] FZI Research Center for Information Technologies at the University of Karlsruhe
Karlsruhe, Germany
motik@fzi.de
[3] Department of Computer Science, University of Manchester
Manchester, UK
sattler@cs.man.ac.uk

Abstract. As applications of description logics proliferate, efficient reasoning with large ABoxes (sets of individuals with descriptions) becomes ever more important. Motivated by the prospects of reusing optimization techniques from deductive databases, in this paper, we present a novel approach to checking consistency of ABoxes, instance checking and query answering, w.r.t. ontologies formulated using a slight restriction of the description logic $\mathcal{SHIQ}$. Our approach proceeds in three steps: (i) the ontology is translated into first-order clauses, (ii) TBox and RBox clauses are saturated using a resolution-based decision procedure, and (iii) the saturated set of clauses is translated into a disjunctive datalog program. Thus, query answering can be performed using the resulting program, while applying all existing optimization techniques, such as join-order optimizations or magic sets. Equally important, the resolution-based decision procedure we present is for unary coding of numbers worst-case optimal, i.e. it runs in ExpTime.

1 Introduction

In recent years description logics have found their application in various fields of computer science, including, but not limiting to data integration, knowledge representation and ontology modeling for the Semantic Web. Many practical DL reasoners have been built and applied to practical problems. The experience shows that such systems perform well when computing the subsumption hierarchy: they use practicable, highly optimized tableau-based algorithms [20], which perform much better on practical problems than their worst-case computational complexity suggests.

However, new applications, such as metadata management in the Semantic Web, require efficient query answering over large ABoxes. So far, attempts have been made to answer queries by reduction to ABox consistency checking, which can already be done by employing above mentioned tableau algorithms. From a theoretical point of

view, this approach is quite elegant, but from a practical point of view, it has a significant drawback: as the number of ABox individuals increases, the performance becomes quite poor. We believe that there are three three main reasons for this. Firstly, tableau-based algorithms treat all individuals separately, i.e., they do not group individuals together depending on common properties. Secondly, 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. We find it difficult to modify the tableau search strategy to take into account the query in the search. Thirdly, a tableau-based ABox algorithm tries to construct a 'forest model', i.e., a model where each individual in the ABox is the root of a tree of (in the worst case) exponential depth. Clearly, an ABox algorithm that is to handle large numbers of ABox individuals has to restrict these trees to a minimum size. These deficiencies have already been acknowledged by the research community, and certain optimization techniques for instance retrieval have already been developed [18, 19]. However, the performance of query answering is still often not satisfactory in practice.

On the other hand, many techniques for efficient reasoning in deductive databases already exist. For example, the first point is addressed by managing individuals in sets [1]. This opens the door to various optimization techniques, such as join order optimization. Consider the query $\mathit{worksAt}(P, I), \mathit{hasName}(I, \text{'FZI'})$. Since the second argument is a constant, it is reasonable to assume that evaluating the subgoal $\mathit{hasName}(I, \text{'FZI'})$ first, and then joining the result with tuples in the *worksAt* relation, will reduce the number of irrelevant tuples considered. Join order optimizations are usually based on the database statistics, and were shown to be very effective in practice.

The second point is related to the observation that in most cases, only a small subset of information from the database actually contributes to the query answer. Hence, by effectively identifying the relevant information subset, query answering can be optimized significantly. Magic sets transformation [8] is a primary technique addressing this problem, and has been used mainly in the context of Horn deductive databases to optimize evaluation of recursive queries. Roughly, the query is modified so that during its evaluation, a set of relevant facts is derived, and checking original query conditions is limited to this estimation. For example, computing ancestors of some person in a genealogy tree requires only considering the nodes from that person upwards. The magic sets transformation for disjunctive programs has been presented recently in [16], along with empirical evidence of its usefulness. The significant performance improvements reported there are attributed mainly to the fact that selecting information relevant to the query reduces the number of models of the disjunctive program.

Since techniques for reasoning in deductive databases are now mature, we think it makes sense to examine how they can be applied to improve ABox reasoning in description logics. Our work was inspired by [17], where the intersection of logic programming and description logic was investigated. Our initial experimental results on ABox reasoning using disjunctive database techniques were reported in [22] and are very promising, exhibiting improvements in performance of one, or sometimes two orders of magnitude. However, the approach presented there was limited to a very simple logic. In particular, the presence of existentials in terminological cycles was not allowed, due to problems

with termination. Other expressive features, such as number restrictions or transitivity, were not considered either.

In this paper, we present a technique for reducing $\mathcal{SHIQ}^-$ knowledge bases to disjunctive datalog programs, while preserving the semantics of the knowledge base. $\mathcal{SHIQ}^-$ is a very expressive description logics which is at the core of Ontology Web Language (OWL) [27] – the current standard for ontology languages in the Semantic Web. $\mathcal{SHIQ}^-$ differs from $\mathcal{SHIQ}$ in the additional restriction that number restrictions are allowed only for roles not having subroles. Our approach for deciding consistency of a $\mathcal{SHIQ}^-$ knowledge base KB works as follows:

- First, we encode KB into an $\mathcal{ALCHIQ}^-$ knowledge base $\Omega(KB)$ to eliminate role transitivity axioms in KB, using an encoding similar to those from [30, 29].
- Next, we translate the TBox and RBox of $\Omega(KB)$ into first-order formulae and transform them into clausal form using structural transformation, obtaining the set of clauses $\Gamma_{\mathcal{TR}}$.
- We then apply the basic superposition calculus to $\Gamma_{\mathcal{TR}}$ to obtain a saturated set of clauses $\mathsf{Sat}(\Gamma_{\mathcal{TR}})$. We choose appropriate ordering and selection function which, together with eager redundancy elimination, ensure termination.
- We translate $\mathsf{Sat}(\Gamma_{\mathcal{TR}})$ into a function-free version $\mathsf{FF}(KB)$, where each ground functional term $f(a)$ is simulated using a new constant a_f.
- Since it does not contain functional terms any more, $\mathsf{FF}(KB)$ is then easily transformed into a disjunctive datalog program with equality $\mathsf{DD}(KB)$.

In Section 4, we show in Lemma 7 that $\mathsf{FF}(KB)$ and KB are equi-satisfiable, and in Theorem 2 that $\mathsf{DD}(KB)$ entails the same set of ground consequences as KB under descriptive semantics. Since $\mathsf{DD}(KB)$ is a positive disjunctive datalog program, any known technique for query answering may be used. In particular, an algorithm for query answering by managing sets of tuples was given in [9], which is based on ordered hyperresolution. Further, estimating the subset of databases information relevant to the query can be achieved by applying the magic sets transformation [16]. Finally, saturating TBox and RBox first allows us to cut down the trees under individuals in the model to the depth of only one.

Complementary to $\mathsf{DD}(KB)$, the saturation-based decision procedure used in the third step to saturate $\Gamma_{\mathcal{TR}}$ can be used for standard TBox reasoning tasks. It is based on the basic superposition calculus by Bachmair and Ganzinger [7]. As demonstrated in Section 3, Theorem 1, basic superposition with eager redundancy elimination decides $\mathcal{ALCHIQ}^-$ satisfiability in worst case exponential time, for unary coding of numbers in the input. This is almost optimal, since $\mathcal{ALCHIQ}^-$ is EXPTIME-complete for binary coding of numbers in the input [30]. Drawing from the vast experience in building efficient theorem provers, our algorithm can be efficiently implemented in practice. On the other hand, existing optimal decision procedures for $\mathcal{ALCHIQ}$ employing tree automata techniques, are known not to be practicable.

We believe that the resolution decision procedure for $\mathcal{ALCHIQ}^-$ is interesting in its own right. Many resolution decision procedures for various classes of logics have already been devised, e.g. for DL* class [25] or for the (loosely) guarded fragment with equality [14]. However, counting quantifiers cannot be embedded in any of these decidable classes. Furthermore, the combination of inverse roles and counting quantifiers is

known to be difficult to handle. On the model-theoretic side, it makes the logic loose the finite model property. On the proof-theoretic side, tableau decision procedures for such logics require sophisticated pair-wise blocking techniques for ensuring termination [20]. Our decision procedure provides an elegant alternative by using so called 'basicness' restriction, combined with eager elimination of redundant clauses by subsumption. To the best of our knowledge, this is the first resolution decision procedure employing redundancy elimination by subsumption to restrict the term depth.

The rest of this paper is structured as follows. In Section 2, we present the preliminaries necessary for understanding our paper. In Section 3, we present the resolution decision procedure for testing satisfiability of $\mathcal{ALCHIQ}^-$ knowledge bases. In Section 4, we show how to reduce an $\mathcal{ALCHIQ}^-$ knowledge base to a disjunctive datalog program. In Section 5, we show how a $\mathcal{SHIQ}$ knowledge base can be reduced to an $\mathcal{ALCHIQ}$ knowledge base, thus extending our results from previous sections. Before we conclude, we present a simple example in Section 6.

2 Preliminaries

2.1 Description Logic $\mathcal{SHIQ}$

In this subsection we introduce the $\mathcal{SHIQ}$ description logic, including its syntax, semantics and interesting inference problems.

Definition 1. *Let N_R be the set of role names. The set of $\mathcal{SHIQ}$* roles *is the set $N_R \cup \{R^- | R \in N_R\}$. For $R \in N_R$, let $\mathsf{Inv}(R)$ denote R^- and let $\mathsf{Inv}(R^-)$ denote R. An* RBox *$\mathcal{R}$ over N_R is a finite set of transitivity axioms $\mathsf{Trans}(R)$ and role inclusion axioms $R \sqsubseteq S$, where R and S are roles, such that, if $R \sqsubseteq S \in \mathcal{R}$, then $\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \in \mathcal{R}$ as well. A role R is* transitive *if $\mathsf{Trans}(R) \in \mathcal{R}$ or $\mathsf{Trans}(\mathsf{Inv}(R)) \in \mathcal{R}$. Let $\sqsubseteq^*$ denote the reflexive-transitive closure of $\sqsubseteq$. A role R is* transitive *if $\mathsf{Trans}(S) \in \mathcal{R}$ or $\mathsf{Trans}(\mathsf{Inv}(S)) \in \mathcal{R}$ for some S with $S \sqsubseteq^* R$ and $R \sqsubseteq^* S$; R is* simple *if there is no role S such that $S \sqsubseteq^* R$ and S is transitive; R is* complex *if it is not simple.*

Let N_C be a set of atomic *concept names. The set of $\mathcal{SHIQ}$* concepts *over N_C and N_R is defined inductively as the smallest set for which the following holds: $\top$ and $\bot$ are $\mathcal{SHIQ}$ concepts, each atomic concept name $A \in N_C$ is a $\mathcal{SHIQ}$ concept, if C and D are $\mathcal{SHIQ}$ concepts and R is a role, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$ are also $\mathcal{SHIQ}$ concepts, and, if C is a $\mathcal{SHIQ}$ concept, R a simple role and n an integer, then $\leq n\, R.C$ and $\geq n\, R.C$ are $\mathcal{SHIQ}$ concepts. Concepts of the latter form, where C is different from $\top$, are called* qualified number restrictions*, whereas concepts of the form $\leq n\, R.\top$ and $\geq n\, R.\top$ are called* unqualified number restrictions *and are written as $\leq n\, R$ and $\geq n\, R$.*

TBox *$\mathcal{T}$ over N_C and $\mathcal{R}$ is a finite set of* concept inclusion axioms $C \sqsubseteq D$ *or* concept equivalence axioms $C \equiv D$*, where C and D are $\mathcal{SHIQ}$ concepts.*

Let N_I be a set of individual names*. An* ABox *$\mathcal{A}$ is a set of concept and role* membership axioms $C(a)$ *and $R(a, b)$, and* (in)equality axioms $a \approx b$ *and $a \not\approx b$, where C is a $\mathcal{SHIQ}$ concept, R a role, and a and b are individuals.*

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

Mapping Concepts to FOL	
$\pi_y(\top, X)$	$= \top$
$\pi_y(\bot, X)$	$= \bot$
$\pi_y(A, X)$	$= A(X)$
$\pi_y(\neg C, X)$	$= \neg\pi_y(C, X)$
$\pi_y(C \sqcap D, X)$	$= \pi_y(C, X) \wedge \pi_y(D, X)$
$\pi_y(C \sqcup D, X)$	$= \pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X)$	$= \forall y : R(X, y) \rightarrow \pi_x(C, y)$
$\pi_y(\exists R.C, X)$	$= \exists y : R(X, y) \wedge \pi_x(C, y)$
$\pi_y(\leq n\, R.C, X)$	$= \forall y_1, \ldots, y_{n+1} : \bigwedge R(X, y_i) \wedge \bigwedge\ \pi_x(C, y_i) \rightarrow \bigvee\ y_i \approx y_j$
$\pi_y(\geq n\, R.C, X)$	$= \exists y_1, \ldots, y_n : \bigwedge R(X, y_i) \wedge \bigwedge\ \pi_x(C, y_i) \wedge \bigwedge\ y_i \not\approx y_j$
Mapping Axioms to FOL	
$\pi(C \sqsubseteq D)$	$= \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$
$\pi(C \equiv D)$	$= \forall x : \pi_y(C, x) \leftrightarrow \pi_y(D, x)$
$\pi(R \sqsubseteq S)$	$= \forall x, y : R(x, y) \rightarrow S(x, y)$
$\pi(\mathsf{Trans}(R))$	$= \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
$\pi(C(a))$	$= \pi_y(C, a)$
$\pi(R(a, b))$	$= R(a, b)$
$\pi(a \approx b)$	$= a \approx b$
$\pi(a \not\approx b)$	$= a \not\approx b$
Mapping KB to FOL	
$\pi(R)$	$= \forall x, y : R(x, y) \leftrightarrow R^-(y, x)$
$\pi(KB_\mathcal{R})$	$= \bigwedge_{\alpha \in KB_\mathcal{R}} \pi(\alpha) \wedge \bigwedge_{R \in N_R} \pi(R)$
$\pi(KB_\mathcal{T})$	$= \bigwedge_{\alpha \in KB_\mathcal{T}} \pi(\alpha)$
$\pi(KB_\mathcal{A})$	$= \bigwedge_{\alpha \in KB_\mathcal{A}} \pi(\alpha)$
$\pi(KB)$	$= \pi(KB_\mathcal{R}) \wedge \pi(KB_\mathcal{T}) \wedge \pi(KB_\mathcal{A})$

Notes:
(*i*): X is a meta variable and is substituted by the actual variable.
(*ii*): π_x is defined as π_y by substituting x and x_i for all y and y_i, respectively.

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

Usually, (in)equality axioms are not allowed in the ABox, but the *unique name assumption* (UNA) is employed, requiring each individual to be interpreted as a different object of the interpretation domain. However, OWL lacks the unique name assumption and enables the user to axiomatize equality and inequality of individuals explicitly. Therefore, we do not incorporate UNA into the definition of $\mathcal{SHIQ}$, but allow the user to axiomatize it explicitly by including an inequality axiom $a_i \not\approx a_j$ for each pair of distinct individuals.

Definition 2. *The semantics of a $\mathcal{SHIQ}$ knowledge base KB is given by the mapping π which transforms KB axioms into first-order formulae, as presented in Table 1. The basic inference problem is checking* satisfiability *of KB, that is, determining whether a first-order model of $\pi(KB)$ exists.*

Other interesting inference problems can be reduced to satisfiability as follows, where α denotes a new individual not occurring in the knowledge base:

- Concept satisfiability. *A concept C is satisfiable with respect to KB if and only if there exists a model of KB in which the interpretation of C is not empty. This is the case if and only if $KB \cup C(\alpha)$ is satisfiable.*
- Subsumption. *A concept C is subsumed by a concept D with respect to KB if and only if $\pi(KB) \models \pi(C \sqsubseteq D)$. This is the case if and only if $KB \cup (C \sqcap \neg D)(\alpha)$ is unsatisfiable.*
- Instance checking. *An individual i is an instance of a concept C with respect to KB if and only if $\pi(KB) \models \pi(C(i))$. This is the case if and only if $KB \cup \neg C(i)$ is unsatisfiable.*

We now define a slight restriction of $\mathcal{SHIQ}$ description logic to which the approach in this paper is applicable.

Definition 3. *For a knowledge base KB, a role R is called* very simple *if no role S different from R exists, such that $S \sqsubseteq^* R \in KB_{\mathcal{R}}$. Description logic $\mathcal{SHIQ}^-$ has the same syntax and semantics as $\mathcal{SHIQ}$, with the additional syntactical restriction that roles in number restrictions $\leq n\, R.C$ and $\geq n\, R.C$ are very simple.*

We also consider the $\mathcal{ALCHIQ}$ ($\mathcal{ALCHIQ}^-$) fragment of $\mathcal{SHIQ}$ ($\mathcal{SHIQ}^-$), which does not allow transitivity axioms.

Notice that $\sqsubseteq^*$ can be a cyclic relation in general. In [30] it has been shown that a $\mathcal{SHIQ}$ knowledge base KB with a cyclic role hierarchy can be reduced to a knowledge base KB' with an acyclic role hierarchy, as follows. First, we compute the set of maximal, strongly connected components (or maximal cycles) of the role inclusion relation $\sqsubseteq$ of KB. For each strongly connected component C, we select one representative role, denoted as $\mathsf{role}(C)$. Next, we form the new TBox $KB'_{\mathcal{T}}$ and ABox $KB'_{\mathcal{A}}$ by replacing, in all axioms of $KB_{\mathcal{A}}$ and $KB_{\mathcal{T}}$, each role R with $\mathsf{role}(C)$, where C is the maximal, strongly connected component that R belongs to. Finally, we construct the new RBox $KB'_{\mathcal{R}}$ as follows: (1) for each pair of strongly connected components $C \neq C'$, we add the axiom $\mathsf{role}(C) \sqsubseteq \mathsf{role}(C')$ to $KB'_{\mathcal{R}}$ if there are roles $R \in C$ and $R' \in C'$ with $R \sqsubseteq^* R'$, and (2) for each strongly connected component C, we add the axiom $\mathsf{Inv}(\mathsf{role}(C)) \sqsubseteq \mathsf{role}(C)$ to $KB'_{\mathcal{R}}$ if there is a role $R \in C$, such that also $\mathsf{Inv}(R) \in C$. Since the strongly connected components of $\sqsubseteq$ can be computed in time quadratic in the number of roles, this reduction can be performed in polynomial time. Hence, we can assume without loss of generality that RBoxes are acyclic.

A concept C is in *negation-normal form* if all negations in it occur in front of atomic concepts only. C can be transformed into an equivalent concept in negation-normal form, denoted as $\mathsf{NNF}(C)$, in time linear in size of C, by repeatedly applying de Morgan's laws to push negation inwards as much as possible.

2.2 Basic Superposition Calculus

Paramodulation is one of the fundamental techniques for theorem proving with equality. In order to improve its performance, in [3] Bachmair and Ganzinger have presented the superposition calculus, where stronger ordering restrictions restrict unnecessary inferences. However, further optimizations of paramodulation and superposition were presented in [7], by adding a so called 'basicness' restriction. These optimizations are very

general, but a simplified version, called basic superposition, may be found in [4, 5]. A very similar calculus, based on an inference model with constrained clauses, was developed by Nieuwenhuis and Rubio [24].

The idea of the basic superposition calculus is to render superposition inferences into terms introduced by previous unification steps redundant. In practice, this technique has been shown essential for solving some particularly difficult problems in first-order logic with equality [21]. Furthermore, it shows that superposition into arguments of Skolem function symbols is not necessary. Namely, any Skolem function symbol f occurs in the original clause set with variable arguments. Hence, for any term $f(t)$, if t is not a variable, it was introduced by a previous unification step.

We assume the standard definition of first-order formulae, which can be found in any text book on first-order theorem proving (e.g. [13]). Clauses are considered to be multisets of literals, denoted as $L_1 \vee \ldots \vee L_n$. This is necessary for correct treatment of simplification rules compatible with the calculus. A clause C is *safe* if each variable occurring in a positive literal of C also occurs in a negative literal of C. *Unsafe* is the opposite of safe. A clause is *positive* if it does not contain a negated literal; it is *negative* if it does not contain positive literals.

We use the following typographic conventions: atoms will be denoted by letters A and B, clauses by C and D, literals by L, predicates by P, R, S, T and U, constants by a, b and c, variables by x, y and z, and terms by s, t, u, v and w. For terms u and s, $u|_p$ denotes the subterm of u at position p, and $u[s]_p$ denotes the term obtained from u by substituting s at position p. Variable assignments of a substitution σ will be written as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, and its set of domain variables as $\mathsf{dom}(\sigma)$. The empty substitution is denoted by $\{\}$, and the result of the applying it to any clause C is C itself.

It is common practice in equational theorem proving to consider logical theories consisting of the equality predicate exclusively. This simplifies the theoretical treatment significantly without losing generality. Literals $P(t_1, \ldots, t_n)$, where P is not the equality predicate, are encoded as $P(t_1, \ldots, t_n) \approx \top$, so predicate symbols actually become function symbols. It is well-known that this transformation preserves satisfiability. In order to avoid considering terms where predicate symbols occur as proper subterms of other terms, one usually employs a multi-sorted framework: all predicate symbols and the symbol $\top$ belong to a sort which is disjoint from the sort of function and constant symbols. In the sequel we treat $P(t_1, \ldots, t_n)$ as a syntactic shortcut. We treat $\approx$ as having built-in symmetry: any literal $s \approx t$ may also be interpreted as $t \approx s$.

As already mentioned, the goal of the calculus is to prohibit inferences into terms introduced by previous unification steps. This is most easily formulated by breaking a clause into two parts: (i) the *skeleton* clause C and (ii) the substitution σ representing the cumulative effects of previous unifications. These two components together are called a *closure* and are written as $C \cdot \sigma$, and are logically equivalent to a clause $C\sigma$. A closure $C \cdot \sigma$ can, for convenience, equivalently be represented as $C\sigma$, where the terms occurring at variable positions of C are *marked*. Any position at or beneath a marked position is called a *substitution position*. Basic superposition can be summarized as a calculus where superposition into a substitution position is not necessary.

The following example is logically equivalent to the clause $P(f(y)) \vee g(b) \approx b$. On the left-hand side, the closure is represented by a skeleton and a substitution explicitly, whereas on the right-hand side it is represented by marking the variable positions[4].

$$(P(x) \vee z \approx b) \cdot \{x \mapsto f(y), z \mapsto g(b)\} \equiv P([f(y)]) \vee [g(b)] \approx b$$

Note that all variable positions are always marked, so we usually do not show this for readability purposes. A closure $C \cdot \sigma$ is *ground* if $C\sigma$ is ground. To technically simplify the presentation, we consider each closure to be in the *standard form*, which is the case if the following conditions are satisfied:

- the substitution σ does not contain trivial mappings of the form $x \mapsto y$ and
- all variables from $\mathsf{dom}(\sigma)$ occur in C.

A closure $C \cdot \sigma$ can be brought into the standard form in the following way: If $x \mapsto t$ is a mapping in σ violating some of the conditions above, then let σ' be $\sigma \setminus \{x \mapsto t\}$, and replace the closure with $C\{x \mapsto t\} \cdot \sigma'\{x \mapsto t\}$.

A closure $(C\sigma_1) \cdot \sigma_2$ is a *retraction* of a closure $C \cdot \sigma$ if $\sigma = \sigma_1\sigma_2$. Intuitively, a retraction is obtained by moving some marked positions lower in the closure. For example, the following is a retraction of the example above:

$$(P(x) \vee g(z) \approx b) \cdot \{x \mapsto f(y), z \mapsto b\} \equiv P([f(y)]) \vee g([b]) \approx b$$

The basic superposition calculus is parameterized with an ordering $\succ$ and a selection function. An *admissible* ordering on terms $\succ$ is a transitive relation satisfying these properties:

- It is well-founded, which means that there is no infinite sequence $t_0 \succ t_1 \succ \ldots$.
- It is a rewrite relation, which means that, if $s \succ t$, then, for any substitution σ and term u, it holds that $u[s\sigma]_p \succ u[t\sigma]_p$.
- It is total on ground terms, which means that for all ground terms t and s, either $s \succ t$, $s = t$ or $t \succ s$.
- $\top$ is the smallest element.

An ordering $\succ$ can be extended to an ordering on literals (ambiguously denoted also with $\succ$) by identifying each positive literal $s \approx t$ with a multiset $\{\{s\}, \{t\}\}$ and each negative literal $s \not\approx t$ with a multiset $\{\{s, t\}\}$, and comparing these multisets by a two-fold multiset extension $(\succ_{mul})_{mul}$ over $\succ$ [5]. An extension $\succ_{mul}$ of some ordering $\succ$ to multisets is defined as follows: $M \succ_{mul} N$ if $M \neq N$ and if $N(x) > M(x)$ for some x, then there is some y for which $M(y) > N(y)$ and $y \succ x$. The literal ordering obtained in such a way is total on ground literals. We say that the literal $L \cdot \sigma$ is *maximal*

[4] In [7], framing was used for marked positions. We decided to use a different notation, because framing introduced problems with the text layout. Our notation should not be confused with the notation for modalities in multi-modal logic.

[5] Whereas this definition should be used in the general case, for the special case of $\mathcal{ALCHIQ}^-$ logic we give a simpler definition in Section 3.

in closure $C \cdot \sigma$ if there is no $L' \in C \setminus \{L\}$ such that $L'\sigma \succ L\sigma$. We say that $L \cdot \sigma$ is *strictly maximal* if there is no $L' \in C \setminus \{L\}$ such that $L'\sigma \succeq L\sigma$.

A *selection function* selects a (possibly empty) subset of negative literals in a closure. There are no other constraints on the selection function.

The basic superposition calculus is a refutation procedure. If some set of closures N is *saturated up to redundancy*, then it is unsatisfiable if and only if it contains the empty closure. Intuitively, the set of closures N is saturated up to redundancy if all inferences from premises in N are redundant in N. The exact details of redundancy will be discussed later; we present the rules of the calculus first. For these rules, we make the technical assumption that all premises are variable-disjoint, so closures in all premises are expressed using the same substitution. A literal $L \cdot \theta$ is *(strictly) eligible for superposition* in the closure $(C \vee L) \cdot \theta$ if there are no selected literals in $(C \vee L) \cdot \theta$ and $L \cdot \theta$ is (strictly) maximal in $C \cdot \theta$. A literal $L \cdot \theta$ is *eligible for resolution* in the closure $(C \vee L) \cdot \theta$ if it is selected in $(C \vee L) \cdot \theta$ or there are no selected literals in $(C \vee L) \cdot \theta$ and $L \cdot \theta$ is maximal in $C \cdot \theta$. We call this calculus $\mathcal{BS}$ and present its inference rules in Table 2. MGU denotes the most general unifier.

In the ordered resolution inference rule, the first closure is called the *side premise* and the second the *main premise*. Ordered resolution is actually a 'macro': negative superposition of $(A \approx \top) \cdot \rho$ from the side premise into $(B \not\approx \top) \cdot \rho$ of the main premise results in $(\top \not\approx \top) \cdot \theta$, which is immediately eliminated by reflexivity resolution. Also, positive superposition of a main premise into a positive literal of the form $(B \approx \top) \cdot \rho$ result in a tautology $(\top \approx \top) \cdot \theta$. Hence, ordered resolution captures all necessary inferences from two premises involving literals with ordinary predicates. One might also define ordered factoring as a macro, combining equality resolution on $(C \vee A \approx \top \vee B \approx \top) \cdot \rho$ with reflexivity resolution. We decided not to do this in order to keep the presentation simple.

As already mentioned, we often use the convenient representation of closures, by marking the position of variables in the skeleton. Inference rules of $\mathcal{BS}$ govern precisely how markers are propagated. For example, consider superposition of $[f(x)] \approx [g(x)]$ into $R(x', f(x'))$. We first represent the premises by showing the skeleton and substitution explicitly: the first premise is equivalent to $(y \approx z) \cdot \{y \mapsto f(x), z \mapsto g(x)\}$, and the second one to $R(x, f(x)) \cdot \{\}$. By the definition of the positive superposition rule in Table 2, the superposition conclusion is clearly $R(x', z) \cdot \{z \mapsto g(x')\}$, which can also be written as $R(x', [g(x')])$.

It is well known that efficient inference rules are just one component of an efficient theorem prover. Equally important are effective *redundancy elimination rules*, providing means for deleting certain closures or replacing them with simpler ones, without jeopardizing completeness. Usual simplification and deletion techniques are not directly applicable in the context of $\mathcal{BS}$. Hence, we present an overview of the techniques for which compatibility with $\mathcal{BS}$ was demonstrated in [7].

The notion of a closure $C \cdot \sigma$ being *reduced modulo substitution η relative to* a closure $D \cdot \theta$ is central in treating simplification rules. However, giving the exact definition of that notion would require presenting the ins and outs of the calculus, so we direct the interested reader to [7]. Intuitively, $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo η if, for all

Positive superposition: $\dfrac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \approx v) \cdot \rho}{(C \vee D \vee w[t]_p \approx v) \cdot \theta}$	(*i*) $\sigma = \text{MGU}(s\rho, w\rho|_p)$ and $\theta = \rho\sigma$, (*ii*) $t\theta \not\succeq s\theta$ and $v\theta \not\succeq w\theta$, (*iii*) $(s \approx t) \cdot \theta$ is strictly eligible for superposition in $(C \vee s \approx t) \cdot \theta$, (*iv*) $(w \approx v) \cdot \theta$ is strictly eligible for superposition in $(D \vee w \approx v) \cdot \theta$, (*v*) $s\theta \approx t\theta \not\succeq w\theta \approx v\theta$, (*vi*) $w|_p$ is not a variable.
Negative superposition: $\dfrac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \not\approx v) \cdot \rho}{(C \vee D \vee w[t]_p \not\approx v) \cdot \theta}$	(*i*) $\sigma = \text{MGU}(s\rho, w\rho|_p)$ and $\theta = \rho\sigma$, (*ii*) $t\theta \not\succeq s\theta$ and $v\theta \not\succeq w\theta$, (*iii*) $(s \approx t) \cdot \theta$ is strictly eligible for superposition in $(C \vee s \approx t) \cdot \theta$, (*iv*) $(w \not\approx v) \cdot \theta$ is eligible for resolution in $(D \vee w \approx v) \cdot \theta$, (*v*) $w|_p$ is not a variable.
Reflexivity resolution: $\dfrac{(C \vee s \not\approx t) \cdot \rho}{C \cdot \theta}$	(*i*) $\sigma = \text{MGU}(s\rho, t\rho)$ and $\theta = \rho\sigma$, (*ii*) $(s \not\approx t) \cdot \theta$ is eligible for resolution in $(C \vee s \not\approx t) \cdot \theta$.
Equality factoring: $\dfrac{(C \vee s \approx t \vee s' \approx t') \cdot \rho}{(C \vee t \not\approx t' \vee s' \approx t') \cdot \theta}$	(*i*) $\sigma = \text{MGU}(s\rho, s'\rho)$ and $\theta = \rho\sigma$, (*ii*) $t\theta \not\succeq s\theta$ and $t'\theta \not\succeq s'\theta$, (*iii*) $(s \approx t) \cdot \theta$ is eligible for superposition in $(C \vee s \approx t \vee s' \approx t') \cdot \theta$.
Ordered resolution: $\dfrac{(C \vee A) \cdot \rho \quad (D \vee \neg B) \cdot \rho}{(C \vee D) \cdot \theta}$	(*i*) $\sigma = \text{MGU}(A\rho, B\rho)$ and $\theta = \rho\sigma$, (*ii*) $A \cdot \theta$ is strictly eligible for superposition in $(C \vee A) \cdot \theta$, (*iii*) $\neg B \cdot \theta$ is eligible for resolution in $(D \vee \neg B) \cdot \theta$.

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

substitutions τ, whenever terms at marked positions of $D \cdot \theta\tau$ cannot be rewritten by some rewrite system R, then no marked term in $C \cdot \sigma\eta\tau$ can be rewritten by R either. Checking this condition is difficult, since one needs to consider all ground substitutions and all possible rewrite systems. However, approximate checks suitable for practice are known.

One of them involves the notion of η-*domination*: for two terms $s \cdot \sigma$ and $t \cdot \theta$, we say that s is η-dominated by t, written $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$, if and only if $s\sigma\eta = t\theta$ and, whenever some variable x from σ occurs in s at position p, then p is in t at or below a variable.

For example, let $s \cdot \sigma = f(g(x), [g(y)])$ and $t \cdot \theta = f([g(c)], [g(h(z))])$. For $\eta = \{x \mapsto c, y \mapsto h(z)\}$, obviously $s\sigma\eta = t\theta$. Furthermore, each marked position from s can be overlaid at or inside a marked position of t, so $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$.

This notion can be extended to literals: $(s \approx t) \cdot \sigma \sqsubseteq_\eta (w \approx v) \cdot \theta$ if and only if $s \cdot \sigma \sqsubseteq_\eta w \cdot \theta$ and $t \cdot \sigma \sqsubseteq_\eta v \cdot \theta$, or $s \cdot \sigma \sqsubseteq_\eta v \cdot \theta$ and $t \cdot \sigma \sqsubseteq_\eta w \cdot \theta$. The definition is

analogous for negative literals, and no literal may η-dominate a literal of the opposite polarity. The extension to closures is performed like this: $C \cdot \sigma \sqsubseteq_\eta D \cdot \theta$ if and only if, for each literal $L_1 \cdot \sigma$ from $C \cdot \sigma$, there exists a distinct literal $L_2 \cdot \theta$ from $D \cdot \theta$, such that $L_1 \cdot \sigma \sqsubseteq_\eta L_2 \cdot \theta$. Note that this definition allows $D \cdot \theta$ to have more literals than $C \cdot \sigma$.

Now if $C \cdot \sigma \sqsubseteq_\eta D \cdot \theta$, then $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo η. It may happen that, for some η, it holds that $L'\sigma\eta = L\theta$, but not $L' \cdot \sigma \sqsubseteq_\eta L \cdot \theta$. One can make $L' \cdot \sigma$ reduced relative to $L \cdot \theta$ by replacing $L' \cdot \sigma$ with a retract $L'' \cdot \sigma'$ in which those substitution positions which L and L' have in common are instantiated. In this way, the application of a simplification or deletion rule may be enabled, while retracting from σ as little information as possible.

With these notions we can finally present the simplification rules. Closure $C \cdot \sigma$ is a *basic subsumer* of $D \cdot \theta$ if there is a substitution η such that $C\sigma\eta \subseteq D\theta$ and $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo η. Additionally, if $C \cdot \sigma$ has fewer literals than $D \cdot \theta$, then $D \cdot \theta$ may be deleted.

A closure $(C \vee A \vee B) \cdot \sigma$ can be replaced with $(C \vee A) \cdot \sigma$ if $A \cdot \sigma \sqsubseteq_{id} B \cdot \sigma$, where id is the identity substitution. This simplification rule is called *duplicate literal deletion*.

A closure $C \cdot \sigma$ can be deleted if $C\sigma$ is a tautology, meaning that $\models C\sigma$. This deletion rule is called *tautology deletion*. Testing whether $C\sigma$ is a tautology requires itself theorem proving. However, the following simple approximate syntactic checks are typically used: $C\sigma$ is a tautology if it contains a pair of literals $(s \approx t) \cdot \sigma$ and $(s' \not\approx t') \cdot \sigma$, such that $s\sigma = s'\sigma$ and $t\sigma = t'\sigma$, or a literal of the form $(s \approx t) \cdot \sigma$ with $s\sigma = t\sigma$.

A closure $(C \vee x \not\approx s) \cdot \sigma$, where $x\sigma \succ s\sigma$ is called a *basic tautology* and can be safely deleted. For example, if $f(x) \succ g(x)$, then the closure $[f(x)] \not\approx g(x)$ is a basic tautology. However, $f(x) \not\approx g(x)$ is not a basic tautology, since $f(x)$ does not occur at a substitution position.

All presented redundancy elimination rules are decidable. In fact, duplicate literal deletion and tautology deletion can be performed in time polynomial in the number of literals. It is well-known that the subsumption check is NP-complete in the number of literals [15], and η-domination can be checked in polynomial time. Finally, the complexity of basic tautology deletion is determined by the complexity of checking ordering constraints.

2.3 Hyperresolution with Superposition and Splitting

We use hyperresolution with superposition and splitting in some proofs in this paper. This calculus was shown to be complete in [3]. It consists of positive and negative superposition rules, reflexivity resolution, equality factoring, and ordered factoring, which are identical to those presented in Subsection 2.2. However, the ordered resolution inference rule is replaced with *hyperresolution*, which resolves several literals at once. In Table 3, we present the new inference rules, hyperresolution and splitting.

The closure $\neg B_1 \vee \ldots \vee \neg B_n \vee D$ in the hyperresolution rule is called the *nucleus*, whereas closures $C_i \vee A_i$ are called *electrons*. Notice that D may, but need not contain negative atoms: one may don't-care non-deterministically hyperresolve any subset of negative literals [6]. On the other hand, electrons are not allowed to contain negative

Hyperresolution: $\dfrac{C_1 \vee A_1 \;\ldots\; C_n \vee A_n \;\; \neg B_1 \vee \ldots \vee \neg B_n \vee D}{C_1\sigma \vee \ldots \vee C_n\sigma \vee D\sigma}$	(*i*) $\sigma = \text{MGU}(A_1, \ldots, A_n, B_1, \ldots, B_n)$, (*ii*) closures C_i do not contain negative literals, (*iii*) A_i is maximal in $C_i \vee A_i$.
Splitting: $\dfrac{N \cup \{C \vee D\}}{N \cup \{C\} \;\mid\; N \cup \{D\}}$	(*i*) N is a set of closures, (*ii*) C and D do not have variables in common.

Table 3. Inference Rules of Hyperresolution with Splitting

literals. Under these considerations, hyperresolution with superposition is a sound and complete inference procedure.

The splitting rule is borrowed from the semantic tableau calculus and represents an explicit case analysis. If some closure consists of two parts not having variables in common, one may separately test the assumption that either part is true. If unsatisfiability is proved in both cases, the initial closure set is evidently unsatisfiable. Each of the cases introduced by the splitting rule is called a *branch*.

2.4 Disjunctive Datalog

In this subsection we briefly present the syntax and semantics of disjunctive datalog. This presentation is standard and may be found in [11, 16].

A *relational schema* R is a finite list of *relation symbols* $(R_1, \ldots, R_k)$, where each relation is of some arity, denoted as $\mathsf{arity}(R_i)$. A *relational database* D over R and a countable domain U is a finite structure $(U, r_1, \ldots, r_k)$ where r_i are finite *relations* over $U^{\mathsf{arity}(R_i)}$. For a relation symbol R_i, relation r_i from a database D is often denoted as $D(R_i)$. D is sometimes also called an *instance* of R.

A *disjunctive datalog program* P is a triple (π, E, I), where E is a relational schema called *extensional schema*, I a relational schema called *intensional schema*, E and I are defined over same domain U, and π is a finite set of rules of the form

$$A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$$

where $n \geq 0$, $m \geq 0$, atoms A_i and B_i are of the form $S(t_1, \ldots, t_n)$ with t_i being a variable or a constant from U. For atoms A_i, S may be from $I \cup \{\approx\}$, whereas for atoms B_i, S may be from $E \cup I \cup \{\approx\}$. For a rule r, the set of atoms $\{A_i\}$ is called the *rule head* and is denoted as $\mathsf{head}(r)$, whereas the set of atoms $\{B_i\}$ is called the *rule body* and is denoted as $\mathsf{body}(r)$. A ground rule with empty body is called a *fact*.

To simplify the treatment, rules are usually required to be *safe*, that is, each variable occurring in a head literal must occur in a body literal as well. In this way the explicit reference to the universe of the program is not needed. Typical definitions of disjunctive datalog program, e.g. from [11, 16], allow negated atoms in the body. This negation,

however, is non-monotonic, and is different from negation in first-order logic. As our approach produces only positive disjunctive datalog programs, we omit negation from our definitions.

The semantics of disjunctive datalog programs is defined as follows. Let P be a disjunctive datalog program and let D be an instance of the extensional schema of P. Then $P_D = P \cup \{S(\mathbf{t}) | \mathbf{t} \in S(D)\}$ denotes a datalog program obtained by adding to P each tuple from D as a fact. The set HU_{P_D} is called the *Herbrand universe* of P_D and contains all constants from P_D. The *ground instance* of P over HU_{P_D}, denoted as $\mathsf{ground}(P, HU_{P_D})$, is the set of ground rules obtained by replacing variables in each rule of P with constants from HU_{P_D} in all possible ways. The *Herbrand base* HB_{P_D} of P_D is the set of all atoms defined by relations from E, I and $\approx$. An *interpretation* M of P_D is a subset of HB. We say that some ground atom A is true in an interpretation M if $A \in M$. Similarly, A is false in M if $A \notin M$. Interpretation M is a *model* of P_D if, for each rule $r \in \mathsf{ground}(P, HU_{P_D})$, if $\mathsf{body}(R) \subseteq M$, then $\mathsf{head}(r) \cap M \neq \emptyset$ and if all atoms from M involving the $\approx$ predicate yield an equality relation. An *equality relation* is a relation that is reflexive, symmetric, transitive, and, for any relation symbol $R \in E \cup I$, if $R(\ldots, a, \ldots) \in M$ and $a \approx b \in M$, then $R(\ldots, b, \ldots) \in M$ as well.

A model M is *minimal* if no subset of M is a model. The semantics of P_D is denoted by $\mathcal{MM}(P)$ and defined to be the set of all minimal models of P_D. Finally, we define the notion of query answering. A ground literal A is a *cautious answer* of P (written $P \models_c A$) if all minimal models of the program contain A; A is a *brave answer* of P (written $P \models_b A$) if at least one minimal model of the program contains A. First-order entailment is analogous to cautious entailment.

3 Deciding $\mathcal{ALCHIQ}^-$ using Basic Superposition

In this section, we present a resolution decision procedure for the $\mathcal{ALCHIQ}^-$ description logic, based on basic superposition calculus. In Section 5, we show how to extend this result to incorporate transitivity axioms.

In the rest of this section, we deal with closures exclusively, for which we often use the convenient notation of marking terms at variable positions of the skeleton. Closures without marked terms should be understood as closures with an empty substitution. For example, $C(x) \vee D(f(x))$ should be understood as the convenient notation for the closure $(C(x) \vee D(f(x))) \cdot \{\}$.

3.1 Overview

Before delving into the details, we present a high-level overview of the technique we use to derive the decision procedure. Given a knowledge base KB, the first step is to preprocess it into a clausal representation, as explained in Subsection 3.2. Let us denote with $\Xi(KB)$ the set of closures derived by preprocessing KB. It is not difficult to see that $\Xi(KB)$ will contain only closures of certain syntactic form, as shown in Table 4.

We denote with $\mathcal{BS}_{DL}$ the $\mathcal{BS}$ calculus, parameterized as described in Definition 6. We apply saturation of $\Xi(KB)$ under $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules next, and denote the obtained closure set by $\mathsf{Sat}(\Xi(KB))$. Since $\mathcal{BS}_{DL}$

is sound and complete [7], $\mathsf{Sat}(\Xi(KB))$ will contain an empty closure if and only if $\Xi(KB)$ is unsatisfiable. In order to obtain a decision procedure, we show that saturation up to redundancy terminates for any $\mathcal{ALCHIQ}^-$ knowledge base KB. This is done in a proof-theoretic way as follows:

- We generalize the types of closures from Table 4 to so called $\mathcal{ALCHIQ}^-$-closures, which are presented in Table 5. In Subsection 3.4, Lemma 1, we show that each closure occurring in $\Xi(KB)$ is an $\mathcal{ALCHIQ}^-$-closure.
- In Subsection 3.4, Lemma 2, we show that, in any $\mathcal{BS}_{DL}$-derivation from $\Xi(KB)$, each inference rule produces either an $\mathcal{ALCHIQ}^-$-closure, or a closure which is redundant (and may be deleted).
- In Subsection 3.5, Lemma 4, we show that, for some finite knowledge base, the set of possible $\mathcal{ALCHIQ}^-$-closures occurring in any $\mathcal{BS}_{DL}$-derivation is finite.
- Termination is now a simple consequence of these two lemmata: in the worst case, one will build the maximal set of $\mathcal{ALCHIQ}^-$-closures, from which all further inferences are redundant. The bound on the size of the set of $\mathcal{ALCHIQ}^-$-closures gives us also the bound on the complexity of the decision procedure, as demonstrated in Theorem 1.

We now define a saturation-based decision procedure for $\mathcal{ALCHIQ}^-$.

Definition 4. *Let KB be some $\mathcal{ALCHIQ}^-$ knowledge base. Let $\mathsf{Consistent}(KB)$ denote an algorithm that transforms KB to $\Xi(KB)$ as defined in Subsection 3.2, applies $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules, and returns 'unsatisfiable' if and only if $\mathsf{Sat}(\Xi(KB))$ contains the empty closure.*

3.2 Preprocessing

The first step in deciding satisfiability of KB is to transform it into clausal form. Straightforward transformation of $\pi(KB)$ into disjunctive normal form has two significant drawbacks. Firstly, the structure of formulae would be destroyed. Secondly, the usual transformation into clausal normal form would increase the size of the closure set exponentially. Hence, we first apply the *structural transformation* [28], also known as *renaming*. Intuitively, for some first-order formula φ, the structural transformation introduces a new name for each subformula of φ. Thus, the original formula structure is preserved. Furthermore, since the number of subformulae of φ is linear in the size of φ, the exponential blowup is avoided.

Let φ be some formula in negation-normal form, and Λ a subset of positions of subformulae of φ. Then $\mathsf{Def}_\Lambda(\varphi)$ is called the *definitional normal form* of φ with respect to Λ and is defined inductively as follows, where p is maximal in $\Lambda \cup \{p\}$ with respect to the prefix ordering on positions, Q_p is a new predicate not occurring in φ, and $x_1, \ldots, x_n$ are the free variables of $\varphi|_p$:

$$\begin{aligned} \mathsf{Def}_\emptyset(\varphi) &= \varphi \\ \mathsf{Def}_{\Lambda\cup\{p\}}(\varphi) &= \mathsf{Def}_\Lambda(\varphi[Q_p(x_1,\ldots,x_n)]_p) \wedge \forall x_1,\ldots,x_n : Q_p(x_1,\ldots,x_n) \rightarrow \varphi|_p \end{aligned}$$

Furthermore, let $\mathsf{Cls}(\varphi)$ denote the set of closures obtained by the usual clausification by structural skolemization [26]. It is well-known [28] that, if φ does not contain equivalences, then $\mathsf{Cls}(\mathsf{Def}_\Lambda(\varphi))$ can be computed in polynomial time. Furthermore, φ is satisfiable if and only if $\mathsf{Cls}(\mathsf{Def}_\Lambda(\varphi))$ satisfiable, where Λ is any set of positions in φ.

Without loss of generality we may assume that all ABox concept membership axioms in KB are expressed using atomic concepts and that all such concepts occur negatively in all TBox axioms: for each membership axiom $C(a)$, where C is either not atomic or occurs positively in some TBox axiom, one may 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 and it preserves the semantics of KB. We call such knowledge bases *extensionally reduced*. In the rest of this paper we assume that all knowledge bases are extensionally reduced.

Definition 5. *For an $\mathcal{ALCHIQ}$ concept C, let $\mathsf{Def}(C) = \mathsf{Def}_\Lambda(\forall x : \pi(\mathsf{NNF}(C), x))$, where Λ denotes the set of positions in $\forall x : \pi(\mathsf{NNF}(C), x)$ corresponding to positions of non-atomic subconcepts of $\mathsf{NNF}(C)$. For an $\mathcal{ALCHIQ}$ knowledge base KB, let $\Xi(KB)$ denote the minimal set of closures satisfying the following conditions:*

- *For each role name $R \in N_R$, $\mathsf{Cls}(\pi(R)) \subseteq \Xi(KB)$.*
- *For each RBox axiom α in KB, $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$.*
- *For each TBox axiom $C \sqsubseteq D$ in KB, $\mathsf{Cls}(\mathsf{Def}(\neg C \sqcup D)) \subseteq \Xi(KB)$.*
- *For each TBox axiom $C \equiv D$ in KB, $\mathsf{Cls}(\mathsf{Def}(\neg C \sqcup D)) \subseteq \Xi(KB)$ and $\mathsf{Cls}(\mathsf{Def}(\neg D \sqcup C)) \subseteq \Xi(KB)$,*
- *For each ABox axiom α in KB, $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$.*

In the above definition, in Λ one can safely omit positions of outer-most disjunctions of $\mathsf{NNF}(C)$, since this reduces the number of closures generated. For example, the negation normal form of the axiom $\neg C \sqcap \neg D \sqsubseteq \exists R.\top$ is $C \sqcup D \sqcup \exists R.\top$, which can be transformed into closure $C(x) \vee D(x) \vee R(x, f(x))$, without introducing a new name for the subconcept $C \sqcup D$.

By definition of π from Table 1, it is easy to see that all closures obtained by this transformation share some common syntactic properties. Table 4 lists the types of closures that $\Xi(KB)$ may contain. Also, since $\mathsf{NNF}(C)$ can be computed in polynomial time, $\Xi(KB)$ can be computed in polynomial time.

3.3 Parameters for Basic Superposition

We use a *lexicographic path ordering* (LPO) [10] to decide $\mathcal{ALCHIQ}^-$. This ordering, denoted as $\succ_{lpo}$, is induced over a precedence $>_P$ over function, constant and predicate symbols. It is well-known that, if $>_P$ is total, then the induced LPO is admissible, and

1	$\neg R(x,y) \vee \mathsf{Inv}(R)(y,x)$
2	$\neg R(x,y) \vee S(x,y)$
3	$\bigvee (\neg) C_i(x) \vee R(x, f(x))$
4	$\bigvee (\neg) C_i(x) \vee (\neg) D(f(x))$
5	$\bigvee (\neg) C_i(x) \vee f_i(x) \not\approx f_j(x)$
6	$\bigvee (\neg) C_i(x)$
7	$\bigvee (\neg) C_i(x) \vee \bigvee_{i=1}^{n} \neg R(x, y_i) \vee \bigvee_{i=1}^{n} D(y_i) \vee \bigvee_{i,j=1;j>i}^{n} y_i \approx y_j$
8	$(\neg) C(a)$
8	$R(a,b)$
10	$a \approx b$
11	$a \not\approx b$

Table 4. Closures Types after Preprocessing

that it has the *subterm property*, that is, for any term t and a non-root position p, we have $t \succ t|_p$. In general, an LPO $\succ_{lpo}$ is defined as follows: $s \succ_{lpo} t$ if

1. t is a variable occurring as a proper subterm of s or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ and
   (a) $f >_P g$ and, for all i with $1 \leq i \leq n$, we have $s \succ_{lpo} t_i$ or
   (b) $f = g$ and, for some j, we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots t_{j-1})$, $s_j \succ_{lpo} t_j$, and $s \succ_{lpo} t_k$ for all k with $j < k \leq n$ or
   (c) $s_j \succeq_{lpo} t$ for some j with $1 \leq j \leq m$.

Definition 6. *We denote with $\mathcal{BS}_{DL}$ the calculus $\mathcal{BS}$ parameterized as follows:*

- *The term ordering $\succ$ is a LPO induced over a total precedence $>_P$ on function, constant and predicate symbols, such that, for any function symbol f, constant symbol c, and predicate symbol p, we have $f >_P c >_P p >_P \top$.*
- *The selection function selects in each closure $C \cdot \sigma$ every negative binary literal*

In $\mathcal{BS}_{DL}$, we need to compare terms and literals only in closures of types 3–6 and 9 from Table 5. It is easy to see that, since LPOs are total on ground terms, and terms in closures of type 3–6 and 9 have at most one variable, any LPO is total on non-ground terms from these closures. In this case, one can use a more direct definition of the literal ordering. We associate with each literal L the triple $c_L = (\max(L), p_L, \min(L))$, where $\max(L)$ is the maximum of the two terms in L, $\min(L)$ is the minimum of the two terms term in L, and p_L is 1 if L is negative, and 0 otherwise. Then $L_1 \succ L_2$ if and only if $c_{L_1} \succ c_{L_2}$, where c_L are compared lexicographically. An LPO is used to compare the first and the third position of c_L, where for the second position we take $1 \succ 0$. It is easy to see that, since $\succ$ is total on terms, this definition is equivalent to the one based on two-fold multiset extension, given in Subsection 2.2.

Ordering and selection constraints from Subsection 2.2 are checked *a posteriori*, that is, after computing the unifier. This is more general, since some terms may be comparable only after unification. For example, $s = f(x)$ and $t = y$ are not comparable using an LPO. However, for $\sigma = \{x \mapsto a, y \mapsto g(f(a))\}$, we have $t\sigma \succ s\sigma$. The

drawback is that the unifier is often computed in vain, just to determine that constraints are not satisfied. However, LPOs are total on terms from closures 3–6 and 9, so we may check ordering and selection constraints *a priory*, that is, before computing the unifier. If s and t be two terms to be compared, they are either both ground or both have the same, single free variable, so they can always be compared before computing σ. Also, if $s \succ t$, then, for any substitution σ, obviously $s\sigma \succ t\sigma$.

3.4 Closure of $\mathcal{ALCHIQ}^-$-closures under Inferences

Next we generalize types of closures from Table 4 to types of so called $\mathcal{ALCHIQ}^-$-closures presented in Table 5. By $\mathbf{P}(x)$ we denote a possibly empty disjunction of the form $(\neg)P_1(x) \vee \ldots \vee (\neg)P_n(x)$. With $\mathbf{P}(\mathbf{f}(x))$ we denote the possibly empty disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_n}(f_n(x))$. Finally, by $\langle t \rangle$ we denote that t may, but need not be marked. In all closure types, some of the disjuncts may be empty. Observe that type 9 allows ground literals of the form $\neg R(a, b)$. This will enable us to use the algorithm for checking whether $(\neg)R(a, b)$ is entailed by KB.

Lemma 1. *Each closure from $\Xi(KB)$ is of exactly one of the types from Table 5. Furthermore, for each function symbol f occurring in $\Xi(KB)$, there is exactly one closure of type 3 containing $f(x)$ unmarked; this closure is called the R^f-*generator*, the disjunction $\mathbf{P^f}(x)$ is called the f-*support*, and R is called the* designated role *for f and is denoted as* $\mathsf{role}(f)$.

Proof. The first claim follows trivially from the definition of $\Xi(KB)$. For the second claim, one should observe that each closure of type 3 is generated by skolemizing an existentially quantified subformula. Since each skolemization requires a fresh function symbol, such a symbol will be associated with exactly one closure of type 3. □

We now prove the main result of this subsection.

Lemma 2. *Let $\Xi(KB) = N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}_{DL}$-derivation, where C is the conclusion derived from premises in N_i. Then C is either an $\mathcal{ALCHIQ}^-$-closure or is redundant.*

Proof. We first prove the property (*max*), determining which literals can be maximal in closures of types 3, 4, 5, 6 and 9 under ordering and selection function as used in $\mathcal{BS}_{DL}$:

- In a closure of type 3, the literal $R(x, \langle f(x) \rangle$ is always maximal.
- In a closure of type 4, the literal $R([f(x)], x)$ is always maximal.
- In a closure of type 5, the literal of the form $(\neg)P(x)$ can be maximal only if the closure does not contain $f(x)$.
- In a closure of type 6, only literals containing the term $f_i([g(x)])$ can be maximal.
- In a closure of type 9, a literal of the form $(\neg)R(a, b)$, $(\neg)P(a)$, $a \approx b$, or $a \not\approx b$ can be maximal only if the closure does not contain a function symbol.

1	$\neg R(x,y) \vee \mathsf{Inv}(R)(y,x)$
2	$\neg R(x,y) \vee S(x,y)$
3	$\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$
4	$\mathbf{P^f}(x) \vee R([f(x)], x)$
5	$\mathbf{P_1}(x) \vee \mathbf{P_2}(\langle \mathbf{f}(x) \rangle) \vee \bigvee [f_i(x)] \approx [f_j(x)] \vee \bigvee \langle f_i(x) \rangle \not\approx \langle f_j(x) \rangle$ (*i*): for each $f_i(x)$ the closure contains $\mathbf{P^{f_i}}(x)$, (*ii*): for each $[f_i(x)] \approx [f_j(x)]$ we have $\mathsf{role}(f_i) = \mathsf{role}(f_j)$.
6	$\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(\langle \mathbf{t} \rangle) \vee \bigvee [t_i] \approx [t_j] \vee \bigvee \langle t_i \rangle \not\approx \langle t_j \rangle$ (*i*): there is at least one term of the form $f_i([g(x)])$, (*ii*): terms t, t_i and t_j are of the form x or $f_i([g(x)])$, (*iii*): for each $f_i([g(x)])$, the closure contains $\mathbf{P^{f_i}}([g(x)])$, (*iv*): the closure contains $\mathbf{P^g}(x)$, (*v*): for each $[f_i(g(x))] \approx [f_j(g(x))]$, we have $\mathsf{role}(f_i) = \mathsf{role}(f_j)$, (*vi*): for each $[f_i(g(x))] \approx x$, there is a closure $P^g(x) \vee \mathsf{role}(f_i)([g(x)], x)$.
7	$\bigvee \neg R([u], y_i) \vee \mathbf{P_1}(\mathbf{y}) \vee \mathbf{P_2}(x) \vee \mathbf{P_3}([\mathbf{f}(x)]) \vee \bigvee [t_i] \approx [t_j] \vee G$ (*i*): there is at least one literal $\neg R([u], y_i)$, (*ii*): terms t_i and t_j are of the form y_i, a constant c, or a term $f([u])$, (*iii*): u is the variable x or u is a constant and x does not appear in the closure, (*iv*): each y_i occurs as the second argument of exactly one $\neg R([u], y_i)$, (*v*): for each pair of variables y_i and y_j, there is a literal $y_i \approx y_j$, (*vi*): G is a closure of type 9, (*vii*): for each $f_i(u)$, the closure contains $\mathbf{P^{f_i}}([u])$, (*viii*): for each $[f_i(u)] \approx [f_j(u)]$, we have $\mathsf{role}(f_i) = \mathsf{role}(f_j)$.
8	$\bigvee \neg R([g(x)], y_i) \vee \mathbf{P_1}(\mathbf{y}) \vee \mathbf{P_2}(x) \vee \mathbf{P_3}([g(x)]) \vee \mathbf{P_4}([\mathbf{f}(g(x))]) \vee \bigvee [t_i] \approx [t_j]$ (*i*): there is at least one literal $\neg R([g(x)], y_i)$, (*ii*): terms t_i and t_j are of the form y_i, x or $f_i([g(x)])$, (*iii*): each y_i occurs as the second argument of exactly one $\neg R([g(x)], y_i)$, (*iv*): for variable y_i, there is a literal $y_i \approx x$, (*v*): for each $f_i([g(x)])$ the closure contains $\mathbf{P^{f_i}}([g(x)])$, (*vi*): the closure contains $\mathbf{P^g}(x)$, (*vii*): for each $[f_i(g(x))] \approx [f_j(g(x))]$, we have $\mathsf{role}(f_i) = \mathsf{role}(f_j)$, (*viii*): for each $[f_i(g(x))] \approx x$, there is a closure $P^g(x) \vee \mathsf{role}(f_i)([g(x)], x)$.
9	$\mathbf{R}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle) \vee \mathbf{P_1}(\langle \mathbf{a} \rangle) \vee \mathbf{P_2}([\mathbf{f}(\mathbf{a})]) \vee \bigvee \langle t_i \rangle \approx \langle t_j \rangle \vee \bigvee \langle t_i \rangle \not\approx \langle t_j \rangle$ (*i*): terms t_i and t_j are of the form a or $f_i([a])$, (*ii*): equality literals may contain only constant terms non-marked, (*iii*): for each $f_i([a])$ the closure contains $\mathbf{P^{f_i}}([a])$, (*iv*): for each $[f_i(a)] \approx [f_j(a)]$, we have $\mathsf{role}(f_i) = \mathsf{role}(f_j)$, (*v*): for each $[f_i(a)] \approx [b]$ there is an $R(a,b)$-*witness* $R(\langle a \rangle, \langle b \rangle) \vee D \cdot \sigma$, where $D \cdot \sigma$ does not contain functional terms, it is contained in this closure, and $R = \mathsf{role}(f_i)$.

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

For closures of type 3 and 4, the claims follow directly from the Definition 6. Furthermore, we may easily see that, for any term t, function symbol f, and predicate symbol P, we have $f(t) \succ P(t)$. For a closure of type 5, we have $P'(f(x)) \succ f(x) \succ P(x)$, so $P(x)$ may be maximal only if the closure does not contain $f(x)$. For a closure of type 6, we have $P''(f(g(x))) \succ f(g(x)) \succ P'(g(x)) \succ g(x) \succ P(x)$, so only literals containing $f(g(x))$ may be maximal. Finally, we may easily see that, for any function symbol f, constants a, b, and c, unary predicate symbol P, and binary predicate symbol R, we have $f(a) \succ P(b)$, $f(a) \succ R(b, c)$ and $f(a) \succ b$. Hence, literals not containing function symbols can be maximal only in closures not containing function symbols.

We next prove the following invariant (*inv*): Each N_i in the derivation contains exactly one R^f-generator for each function symbol f. The proof is by an easy induction argument. The induction base for N_0 follows directly from Lemma 1. For the induction step, we have to consider all inference rules that can be applied to premises in N_i to a closure of type 3. For positive or negative superposition, it is enough to observe that functional terms always occur marked in an equality literal. Hence, the superposition conclusion always contains only marked functional terms. Ordered resolution of a closure of type 3 is possible only with a closure of type 1 or 2, and results in a closure of type 4 or 3, respectively. However, the term $f(x)$ in both cases obviously occurs marked in the conclusion. Hence, none of these inferences produces an R^f-generator. Since no other inference may produce a closure of type 3, the invariant holds.

In a similar way one may show that no $R(a, b)$-witness contains a negative binary literal. Namely, if a closure C of type 9 contains a negative binary literal, this literal is always selected in C. Hence, C cannot be resolved with a closure of type 7, so it cannot become a witness.

We now prove the Lemma by induction on the derivation length. By Lemma 1, N_0 contains only $\mathcal{ALCHIQ}^-$-closures, so the induction base holds. For the induction step, we examine all possible applications of inference rules of $\mathcal{BS}_{DL}$ to closures in N_i.

Positive or negative superposition. Superposition from a closure of type 7 or 8 is not possible, since these always contain a negative literal $\neg R([u], y_i)$ or $\neg R([g(x)], y_i)$ which is selected. Similarly, superposition into these closures is redundant: u is the only non-variable term occurring in a selected literal and is always marked. Similarly, superposition into closures of type 1, 2, or 4 is redundant, as these closures contain only variables or marked terms at all candidate positions.

Assume $(D \vee w \approx v) \cdot \rho$ is of type 3 with the free variable being x'. By condition (*max*), $(w \approx v) \cdot \rho$ can only be the literal $R(x', f(x'))$ with R being the designated role for f. There are three possibilities:

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 5 with $(s \approx t) \cdot \rho$ of the form $[f(x)] \approx [g(x)]$. Then the unifier σ is $\{x' \mapsto x\}$, so the superposition conclusion has the form $\mathbf{P^f}(x) \vee R(x, [g(x)]) \vee C \cdot \rho$, where $C \cdot \rho$ contains $\mathbf{P^g}(x)$. By condition 5.ii and (*inv*), the R^g-generator of the form $P^g(y) \vee R(y, g(y))$ exists, and subsumes the conclusion through the substitution $\{y \mapsto x\}$.
- Assume that $(C \vee s \approx t) \cdot \rho$ is a closure of type 6. There are two possibilities for the form of $(s \approx t) \cdot \rho$:

  - For superposition from $[f(g(x))] \approx [h(g(x))]$, σ is $\{x' \mapsto g(x)\}$, so the superposition conclusion has the form $\mathbf{P^f}([g(x)]) \vee R([g(x)], [h(g(x))]) \vee C \cdot \rho$, where $C \cdot \rho$ contains $\mathbf{P^h}(g(x))$. By condition 6.v and (*inv*), the R^h-generator of the form $P^h(y) \vee R(y, h(y))$ exists, and subsumes the conclusion through the substitution $\{y \mapsto g(x)\}$.
  - For superposition from $[f(g(x))] \approx x$, σ is $\{x' \mapsto g(x)\}$, so the superposition conclusion has the form $\mathbf{P^f}([g(x)]) \vee R([g(x)], x) \vee C \cdot \rho$, where $C \cdot \rho$ contains $\mathbf{P^g}(x)$. By condition 6.vi and (*inv*), a closure $\mathbf{P^g}(y) \vee R([g(y)], y)$ exists, and obviously subsumes the conclusion through the substitution $\{y \mapsto x\}$.

– $(C \vee s \approx t) \cdot \rho$ is a closure of type 9. There are two possibilities for the form of $(s \approx t) \cdot \rho$:
  - For superposition from $[f(a)] \approx [g(a)]$, σ is $\{x' \mapsto a\}$, so the superposition conclusion has the form $\mathbf{P^f}([a]) \vee R([a], [g(a)]) \vee C \cdot \rho$, where $C \cdot \rho$ contains $\mathbf{P^g}(a)$. By condition 9.iv and (*inv*), the R^g-generator $P^g(y) \vee R(y, g(y))$ exists, and subsumes the conclusion through the substitution $\{y \mapsto a\}$.
  - For superposition from $[f(a)] \approx [b]$, σ is $\{x' \mapsto a\}$, so the superposition conclusion has the form $\mathbf{P^f}([a]) \vee R([a], [b]) \vee C \cdot \rho$. By condition 9.v, an $R(a, b)$-witness of the form $R(\langle a \rangle, \langle b \rangle) \vee D \cdot \sigma$, where $D\sigma \subseteq C\rho$, and R is the designated role for f exists, and subsumes the superposition conclusion through an empty substitution. If the $R(a, b)$-witness has been subsumed by some other closure, then, since the subsumption relation is transitive, this other closure subsumes the conclusion.

In all cases, the superposition conclusion is subsumed by an existing closure, so we may conclude that any superposition into a closure of type 3 is redundant.

Assume $(D \vee w \approx v) \cdot \rho$ is of type 5 with the free variable being x'. By condition (*max*), superposition may be performed into literals $P_2(f(x'))$ or $f(x') \not\approx \ldots$. There are three possibilities:

– $(C \vee s \approx t) \cdot \rho$ is a closure of type 5 with $(s \approx t) \cdot \rho$ of the form $[f(x)] \approx [h(x)]$, and σ is $\{x' \mapsto x\}$. If both premises satisfy conditions 5.i and 5.ii, the conclusion obviously satisfies them too, so it is of type 5.

– $(C \vee s \approx t) \cdot \rho$ is a closure of type 6. Regardless of whether $(s \approx t) \cdot \rho$ is of the form $[f(g(x))] \approx [h(g(x))]$ or $[f(g(x))] \approx x$, the unifier σ is $\{x' \mapsto g(x)\}$. If the premises satisfy conditions 5.i, 6.iii and 6.iv, the conclusion satisfies them too: $\mathbf{P^g}(x)$ is contained in $C \cdot \rho$, disjunctions $\mathbf{P^f}([g(x)])$ from $C \cdot \rho$ are contained in the conclusion, and each $\mathbf{P^f}(x')$ from $D \cdot \rho$ becomes $\mathbf{P^f}([g(x)])$. If both premises satisfy conditions 5.ii, 6.v and 6.vi, the conclusion satisfies them too: literals $[f_i(x')] \approx [f_j(x')]$ from $D \cdot \rho$ satisfying 5.ii are changed into literals $[f_i(g(x))] \approx [f_j(g(x))]$ and satisfy 6.v. No new literals of the form $[f_i(g(x))] \approx x$ are generated, so the conclusion satisfies 6.vi. Hence, the conclusion is of type 6.

– $(C \vee s \approx t) \cdot \rho$ is a closure of type 9 with $(s \approx t) \cdot \rho$ of the form $[f(a)] \approx [b]$ or $[f(a)] \approx [h(a)]$, and the unifier σ is $\{x' \mapsto a\}$. Furthermore, all equalities in $D \cdot \rho$ are of the form $[f_i(x')] \approx [f_j(x')]$, so condition 9.v is satisfied. All $\mathbf{P^f}(\mathbf{x}')$ from $D \cdot \rho$ become $\mathbf{P^f}([\mathbf{a}])$, so condition 9.iii is satisfied. Finally, each $[f_i(x')] \approx [f_j(x')]$

from $D \cdot \rho$ satisfying 5.ii is changed into $[f_i(a)] \approx [f_j(a)]$ and satisfies 9.iv. All equalities in $D \cdot \rho$ have marked terms, so the conclusion satisfies 9.ii. No new equalities of the form $[f_i(a)] \approx [b]$ are generated, so the conclusion satisfies 9.v. Hence, the conclusion is of type 9.

Assume $(D \vee w \approx v) \cdot \rho$ is of type 6 with the free variable x'. By condition (*max*), superposition may be performed into a literal of the form $P_3(f([g(x')]))$ or $f([g(x')]) \not\approx \ldots$. It is important to notice that each occurrence of $g(x)$ in such a closure is always marked, so superposition is possible only at the position of f. There are three possibilities:

- $(C \vee s \approx t) \cdot \rho$ is of type 5 with $(s \approx t) \cdot \rho$ being $[f(x)] \approx [h(x)]$, and the unifier σ is $\{x \mapsto g(x')\}$. If both premises satisfy conditions 5.i, 6.iii, and 6.iv, the conclusion satisfies them too: $\mathbf{P^g}(x')$ is contained in $D \cdot \rho$, each $\mathbf{P^f}([g(x')])$ from $D \cdot \rho$ is contained in the conclusion, and each $\mathbf{P^f}(x)$ from $C \cdot \rho$ becomes $\mathbf{P^f}([g(x')])$. If both premises satisfy conditions 5.ii, 6.v, and 6.vi, the conclusion satisfies them too: literals $[f_i(x)] \approx [f_j(x)]$ from $C \cdot \rho$ satisfying 5.ii are changed into literals $[f_i(g(x'))] \approx [f_j(g(x'))]$ and satisfy 6.v. No new equalities of the form $[f_i(g(x))] \approx x$ are generated, so 6.vi is satisfied, and the conclusion is of type 6.
- $(C \vee s \approx t) \cdot \rho$ is a closure of type 6. Regardless of whether $(s \approx t) \cdot \rho$ has the form $[f(g(x))] \approx [h(g(x))]$ or $[f(g(x))] \approx x$, the unifier σ is $\{x' \mapsto x\}$. Now if both literals involved in the inference are of the form $[f(g(x))] \approx x$, the conclusion contains the literal $x \not\approx x$ which may be eliminated using equality resolution. Hence, the conclusion has the form 5 or 6. Otherwise, at least one of the literals is of the form $[f(g(x))] \approx [h(g(x))]$ and the conclusion has the form 6. In any case, all conditions obviously remain preserved, since the unifier is empty, so each equality in the conclusion must be contained in some premise.
- $(C \vee s \approx t) \cdot \rho$ is a closure of type 9 with $(s \approx t) \cdot \rho$ of the form $[f(a)] \approx [b]$ or $[f(a)] \approx [g(a)]$. However, unification with a term $f([g(x)])$ is not possible, so this type of superposition is not possible.

Assume $(D \vee w \approx v) \cdot \rho$ is of type 9. By condition (*max*), superposition may be performed into a literal of the form $P_1(a)$, $P_2(f([a]))$, $R(a, b)$, $a \not\approx \ldots$ or $f([a]) \not\approx \ldots$. There are three possibilities:

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 5 with $(s \approx t) \cdot \rho$ of the form $[f(x)] \approx [g(x)]$, and the unifier σ is $\{x \mapsto a\}$. Furthermore, all equalities in the closure of type 5 are of the form $[f(x)] \approx [g(x)]$, so condition 9.v remains preserved. All $\mathbf{P^f}(\mathbf{x})$ from $C \cdot \rho$ become $\mathbf{P^f}([\mathbf{a}])$, so condition 9.iii is satisfied. Finally, each $[f_i(x)] \approx [f_j(x)]$ from $C \cdot \rho$ satisfying 5.ii is changed into $[f_i(a)] \approx [f_j(a)]$ and satisfies 9.iv. Condition 9.ii is trivially satisfied. Hence, the conclusion is of type 9.
- $(C \vee s \approx t) \cdot \rho$ is a closure of type 6. Regardless of whether $(s \approx t) \cdot \rho$ has the form $[f(g(x))] \approx [h(g(x))]$ or $[f(g(x))] \approx x$, the unification is not possible.

– $(C \vee s \approx t) \cdot \rho$ is a closure of type 9. Unifier is always empty. Since according to condition 9.ii, all literals of the form $[f(a)] \approx [b]$ have both terms marked, no such literal can be involved in superposition. Hence, condition 9.v remains preserved in the conclusion. Since unifier is empty, no new equalities are generated, so the conclusion obviously satisfies 9.ii, 9.iii and 9.iv. Hence, the conclusion is of type 9.

Reflexivity resolution. Only closures of types 5, 6 or 9 are candidates for reflexivity resolution. For type 5, the unifier is empty, so the conclusion is of type 5.

For type 6, the rule can be applied only to a literal $\langle f([g(x)])\rangle \not\approx \langle f([g(x)])\rangle$ (for literals $\langle f([g(x)])\rangle \not\approx x$ or $\langle f_i([g(x)])\rangle \not\approx \langle f_j([g(x)])\rangle$ the unifier does not exist). The conclusion is of type 5 or 6, depending on whether remaining literals in the closure contain a term of the form $f([g(x)])$.

For type 9, the rule can be applied only to a literal $\langle f([a])\rangle \not\approx \langle f([a])\rangle$ or $\langle a\rangle \not\approx \langle a\rangle$. The unifier is empty and the conclusion is of type 9.

In each case the unifier is empty, so the conclusion obviously satisfies all conditions of the respective closure type. Furthermore, the conclusion always subsumes the premise, so reflexivity resolution can be applied eagerly as a simplification rule.

Equality factoring. Only closures of types 5, 6 or 9 are candidates for equality factoring. The premise has the form $(C \vee s \approx t \vee s' \approx t') \cdot \rho$, where $s\rho \approx t\rho$ is maximal in $C\rho \vee s'\rho \approx t'\rho$, $t\rho \not\succeq s\rho$, and $t'\rho \not\succeq s'\rho$. The unifier σ is always empty. If we assume that simplification by duplicate literal elimination is applied eagerly, we safely conclude that $(s \approx t) \cdot \rho$ is actually strictly maximal, so t' and t cannot be $\top$. Hence, terms $s\rho$, $t\rho$, $s'\rho$ and $t'\rho$ are either all ground or all contain the same free variable x. The ordering $\succ$ is total on such terms, so we may rewrite ordering constraints as $t\rho \prec s\rho$ and $t'\rho \prec s'\rho$. By the fact that $s\rho \approx t\rho$ is strictly maximal, we conclude that $t\rho \succ t'\rho$.

Consider now the case where all equalities involved in the inference are marked, so s, t, s' and t' are variables. This is the case for all closures of type 5 and 6, and some closures of type 9. The conclusion has then the form $(C \vee t \not\approx t' \vee s' \approx t') \cdot \rho$, where t is a variable and $t\rho \succ t'\rho$. Thus, the conclusion is a basic tautology and is redundant.

Hence, provided that duplicate literal elimination is applied eagerly, equality factoring is redundant for all closures, apart from those closures of type 9 containing equalities of the form $\langle a\rangle \approx \langle b\rangle$ with at least one non-marked term, originating from explicit equality statements among individuals. Depending on the marking, equality factoring either yields a basic tautology which is redundant or closure of type 9. In the latter case, the unifier is empty, so the conclusion obviously satisfies all conditions for type 9.

Ordered resolution. Assume that the main premise is a closure of type 1. Then if the side premise is a closure of type 3, the conclusion is a closure of type 4. If the side premise is a closure of type 4, the inference is redundant, since the conclusion is a closure of type 3 which already exists in the closure set. Finally, if the side premise is a closure of type 9, the conclusion is a closure of type 9: conditions 9.ii, 9.iii and 9.iv are obviously preserved, as well as the condition 9.v: the set of witness closures of the conclusion is identical to the set of witnesses of the premise.

Similarly, assume that the main premise is a closure of type 2. Then if the side premise is a closure of type 3, the conclusion is a closure of type 3, but with the term

$f(x)$ marked, so the uniqueness of the designated role for f remains preserved. If the side premise is a closure of type 4, the conclusion is a closure of type 4. Finally, if the side premise is a closure of type 9, the conclusion is a closure of type 9, similarly to the previous case.

Closures of type 3 or 4 cannot be main premises, since their maximal literals are always positive and no negative literals are ever selected.

Consider a resolution between closures of type 5 and 5, 5 and 6 or 6 and 6. By condition (*max*), if some literal $P_1(x)$ is maximal in a closure of type 5, then this closure does not contain the term $f(x)$, and in a closure of type 6 the maximal literal is always of the form $P_3(\langle f([g(x)])\rangle)$. Now in any resolution the unifier is either $\{x' \mapsto x\}$, $\{x' \mapsto g(x)\}$ or $\{x' \mapsto f(g(x))\}$. The conclusion is obviously a closure of type 5 or 6. If either closure contains $f(x)$ ($f_i(g(x))$), then no inference may be performed on some literal from $\mathbf{P^f}(x)$ ($\mathbf{P^{f_i}}([g(x)])$), since all such literals are not maximal. Therefore, the conclusion satisfies conditions 5.i, 6.iii and 6.iv. It can be shown that the conclusion satisfies conditions 5.ii, 6.v, and 6.vi, similarly to the superposition case.

Similarly, resolution between a closure of type 5 and of type 9 results in a closure of type 9. If the closure of type 9 contains $[f(a)] \approx [b]$, resolution on some literal occurring in an $R(a,b)$-witness is not possible by condition (*max*), so the conclusion satisfies 9.v. If some premise contains $f_i(x)$, then resolution on some literal occurring in $\mathbf{P^{f_i}}(x)$ is not possible, so the conclusion satisfies 9.iii. Since both premises satisfy 5.ii, 9.ii and 9.iv, the conclusion satisfies 9.ii and 9.iv as well.

Resolving closures of type 6 and 9 is not possible since maximal literals in the closures do not unify. Finally, closures of type 9 cannot be resolved with closures of type 3 or 4, since literals $\neg R(a,b)$ do not unify with $R(x, \langle f(x)\rangle)$ or $R([f(x)], x)$.

Resolution between two closures of type 9 obviously results in a closure of type 9 or an empty closure: if a premise contains $[f(a)] \approx [b]$, resolution on some literal occurring in its $R(a,b)$-witness is not possible by condition (*max*), so conclusion satisfies 9.v. Since premises satisfy 9.ii, 9.iii, and 9.iv, the conclusion satisfies 9.ii, 9.iii, and 9.iv as well.

Assume that the main premise is a closure of type 7. Resolution is possible only on a selected literal $\neg R([u], y_i')$. There are three possibilities:

- The side premise is a closure of type 3. The unifier σ is $\{x \mapsto u, y_i' \mapsto f(u)\}$ and the conclusion is obviously a closure of type 5, 7, or 9, depending on whether there is another literal $\neg R([u], y_j')$ or not, and whether u is a constant or a variable. If u is a constant a, condition 9.v remains preserved: if there is some $b \approx y_i$ in the main premise, it was produced by resolving it with some closure of type 9 containing the literal $R(a,b)$, which is the $R(a,b)$-witness of the conclusion. All equalities in the conclusion of the form $[f_i(x)] \approx [f_j(x)]$ are generated by resolution with generators involving R. Since all such R are very simple roles, the conclusion satisfies conditions 5.ii, 7.viii or 9.iv. If the conclusion is of type 7, since the main premise satisfies 7.v, the conclusion satisfies 7.v as well. The other conditions are trivial to check.

- The side premise is a closure of type 4. Unification is possible only if u is a variable x'. The unifier σ is $\{x' \mapsto g(x), y_i' \mapsto x\}$ and the conclusion is a closure of type 6

or 8. All equalities $[f_i(g(x))] \approx [f_j(g(x))]$ are obtained from corresponding equalities of the form $[f_i(x)] \approx [f_j(x)]$ from the main premise satisfying 7.viii, so the conclusion satisfies 6.v or 8.vii. All equalities containing $[f_i(g(x))]$ are produced by resolution with the generators involving R. Since all such R are very simple roles, the conclusion obviously satisfies conditions 6.vi or 8.viii. If the conclusion is of type 8, since the main premise satisfies 7.v, the conclusion satisfies 8.iv.

– The side premise is a closure of type 9 with the maximal literal having the form $R(a, b)$. Since the maximal literal in the side premise does not contain function symbols, the entire side premise does not contain function symbols. If u is a variable x', then $\sigma = \{x' \mapsto a, y'_i \mapsto b\}$, otherwise, unification is possible only if u is a constant a and $\sigma = \{y'_i \mapsto b\}$. Application of the unifier may only instantiate new equalities of the form $[f(a)] \approx [g(a)]$, $[f(a)] \approx [b]$ or $\langle a \rangle \approx \langle b \rangle$. Hence, condition 9.v is satisfied in the conclusion – the $R(a, b)$-witness is the side premise. The conclusion is either a closure of type 7 or 9, depending on whether the side premise contains other literals of the form $\neg R([u], y_i)$ or not. Since the main premise satisfies 7.viii and all equalities of the form $[f(a)] \approx [g(a)]$ are inherited from the main premise, the conclusion obviously satisfies 7.viii or 9.iv. If the conclusion is of type 7, since the main premise satisfies 7.v, the conclusion satisfies 7.v as well.

Finally, assume that the main premise is a closure of type 8. Resolution is possible only on selected literals of the form $\neg R([g(x')], y_i)$. There are three possibilities:

– The side premise is a closure of type 3, σ is $\{x \mapsto g(x'), y'_i \mapsto f(g(x'))\}$, and the conclusion is obviously a closure of type 6 or 8: all equalities in the conclusion of the form $[f_i(g(x))] \approx [f_j(g(x))]$ are generated by resolution with generators involving R. Since all such R are very simple roles, the conclusion obviously satisfies conditions 6.v and 6.vi or 8.vii and 8.viii. If the conclusion is of type 8, since the main premise satisfies 8.iv, the conclusion satisfies it as well. The other conditions are trivial to check.

– The side premise is a closure of type 4. Resolution is possible only on the literal of the form $R([g(x)], x)$, with the unifier $\{y'_i \mapsto x\}$. By condition 8.iv, the main premise contains $y'_i \approx x$. Hence, the conclusion contains a literal of the form $x \approx x$, and is a tautology, so this inference is redundant.

– The side is a closure of type 9 with the maximal literal $R(a, b)$. However, unification with a literal $\neg R([g(x')], y'_i)$ is not possible.

We have shown that, for all possible inferences from premises in N_i, the conclusion is either an $\mathcal{ALCHIQ}^-$-closure, or is redundant, so the claim of the Lemma follows. □

A slight optimization of the above process is possible. Namely, any closure of type 7 with n binary literals can be resolved with n premises in $n!$ ways. However, closures of type 7 in $\Xi(KB)$ are symmetric with respect to variables y_i, so all of the $n!$ resolutions will result in the same closure. Obviously, this can be optimized by ordering the premises and performing just one resolution.

We formalize this idea by attaching a constraint $T = y_1 \succ \ldots \succ y_n$ to closures of type 7 in $\Xi(KB)$ and resolving binary literals in closures of type 7 and 8 from left to right. Each time a closure of type 7 participates in a resolution with unifier σ, we compute $T\sigma$. If ordering constraints are not satisfied, the resulting closure may be deleted; otherwise, we attach the constraint $T\sigma$ to the result.

Lemma 3. *The constraint inheritance explained above does not affect soundness or completeness of $\mathcal{BS}_{DL}$.*

3.5 Termination and Complexity Analysis

We now show that the number of $\mathcal{ALCHIQ}^-$-closures is finite for some finite signature. This, in combination with Lemma 2 and the soundness and completeness of $\mathcal{BS}_{DL}$ shows that $\mathcal{BS}_{DL}$, with eager application of redundancy elimination rules, is a decision procedure for $\mathcal{ALCHIQ}^-$.

Let the number $|KB|$ be the *size of the knowledge base*, computed recursively in the following way, where C and D are concepts, A an atomic concept, and R and S roles:

- $|KB| = \sum_{\alpha \in KB_{\mathcal{R}} \cup KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} |\alpha|$,
- $|R \sqsubseteq S| = 3$,
- $|\mathsf{Trans}(R)| = 2$,
- $|C \sqsubseteq D| = |C \equiv D| = |C| + |D| + 1$,
- $|R(a, b)| = 3$,
- $|C(a)| = |C| + 1$,
- $|\top| = |\bot| = 1$,
- $|A| = |\neg A| = 2$,
- $|C \sqcup D| = |C \sqcap D| = |C| + |D| + 1$,
- $|\exists R.C| = |\forall R.C| = 2 + |C|$,
- $| \geq n\, R.C| = | \leq n\, R.C| = n + 2 + |C|$.

Intuitively, $|KB|$ is the number of symbols needed to encode KB on the input tape of a Turing machine using the unary encoding of numbers. We use a single symbol for each atomic concept, role and individual. The n in the definition of the length of concepts $\geq n\, R.C$ and $\leq n\, R.C$ stems from the assumption on unary coding of numbers: a number n can be encoded in unary coding with n bits.

Lemma 4. *Let N_i be any closure set obtained in a derivation as defined in Lemma 2. If C is a closure in N_i, then the number of literals in C is at most polynomial in $|KB|$, for unary coding of numbers in KB input. Furthermore, $|N_i|$ is at most exponential in $|KB|$, for unary coding of numbers in KB input.*

Proof. By Lemma 2, N_i can contain only $\mathcal{ALCHIQ}^-$-closures. Since redundancy elimination is applied eagerly, N_i cannot contain closures with duplicate literals or closures identical up to variable renaming. Let r denote the number of role predicate names, c the number of concept predicate names, i the number of individual names and f the number of function symbols occurring in the signature of $\Xi(KB)$. Then r and i are obviously linear in $|KB|$. Furthermore, c is also linear in $|KB|$, since the number

of new concept names introduced during preprocessing is bounded by the number of subconcepts of each concept, which is linear in $|KB|$. The number f is bounded by the sum of all numbers n in $\geq n\,R.C$ and $\leq n\,R.C$ plus one for each $\exists R.C$ and $\forall R.C$ occurring in KB. Since unary coding of numbers is employed, f is linear in $|KB|$. Let n denote the maximal number occurring in any number restriction. For unary coding of numbers, n is linear in $|KB|$.

Consider now the maximal number of literals in a closure of type 5. The maximal number of literals for $\mathbf{P_1}(x)$ is $2c$ (factor 2 allows for each predicate to occur positively or negatively), for $\mathbf{P_2}(\langle \mathbf{f}(x) \rangle)$ it is $2c \cdot 2f$ (f is multiplied by 2 since each term may or may not be marked), for equalities it is f^2 (both terms are always marked), and for inequalities it is $4f^2$ (factor 4 allows for each side of the equality to be marked or not). Hence, the maximal number of literals is $2c + 4cf + f^2 + 4f^2$. For a closure of type 6, the maximal number of literals is $2c + 2c + 4cf + (f^2 + f) + (4f^2 + 2f)$: possible choices for g do not contribute to the closure length, and the expressions in parenthesis take into account that each term in an equality or an inequality can be $f_i(g(x))$ or x. For a closure of type 9, the maximal number of literals is $2r \cdot 2i \cdot 2i + 2c \cdot 2i + 2c \cdot 2f \cdot i + 2 \cdot (4i^2 + i \cdot f^2 + if \cdot 2i)$: the factor 2 in front of the parenthesis takes into account that equalities and inequalities may have the same form, and the expression in the parenthesis counts all possible forms these literals may have. The maximal number of literals of closures of type 1 and 2 is obviously 2, and for closures of type 3 and 4 it is $c + 1$. For closures of type 7 and 8, one may observe that the number of variables y_i is bounded by n: closures in $\Xi(KB)$ contain at most n variables, and no inference steps increases the number of variables. The maximal number of literals in a closure of type 7 is $n + nc + c + cf + n^2 + |G|$, where $|G|$ is the maximal number of literals in a closure of type 9: choices for u and R do not contribute to the closure length, and the maximal number of equalities is bounded by n^2, since after translation of KB the closure contains a literal $y_i \approx y_j$ for each $i > j$, and no inference increases the number of such equalities. Similarly, the maximal number of literals in a closure of type 8 is $n + nc + c + c + fc + n^2$. Hence, the maximal number of literals in any closure is polynomial in $|KB|$, for unary coding of numbers.

The maximal number of closures of type 1–6 and 9 in N_i is now easily obtained as follows: if C_l is the closure with maximal number of literals l for some closure type, then there are 2^l subsets of literals of C_l. To obtain the total number of closures, one must multiply 2^l with the number of closure-wide choices. For closures of type 6, the function symbol g can be chosen in f ways. For closures of type 3 and 4, one can choose R and f in rf ways. For closures of type 1, one can choose R in r ways. For closures of type 2, one can choose R and S in r^2 ways. Since all these factors are polynomial in $|KB|$ for unary coding of numbers, we obtain an exponential bound on the number of closures of types 1–6 and 9.

For closures of type 7 and 8, we obtain the bound slightly differently: the initial number of closures of type 7 is polynomial in $|KB|$. Each such closure can participate in at most n resolutions on the negative binary literals with a closure of type 3, 4, or 9. Since the number of such closures is exponential in $|KB|$, the number of closures of type 7 and 8 is exponential in $n \cdot |KB|$, which is again exponential in $|KB|$. □

We note that using binary coding of numbers, it is possible to encode the number n by $\log_2 n$ bits, so f and n would be bounded by $2^{|KB|}$, which would yield an exponential bound on the number of literals in the closure, and a doubly-exponential bound on the number of closures.

Theorem 1. *For an $\mathcal{ALCHIQ}^-$ knowledge base KB, $\mathsf{Consistent}(KB)$ decides satisfiability of KB and runs in time exponential in the size of the input for unary coding of numbers.*

Proof. Translation of KB to $\Xi(KB)$ can be performed in time polynomial in the size of KB and contains only $\mathcal{ALCHIQ}^-$-closures by Lemma 1. Let c denote the maximal number of closures occurring in the closure set in a derivation as specified in Lemma 2, and let l denote the maximal number of literals in a closure. By Lemma 4, c is exponential, and l polynomial in $|KB|$, for unary coding of numbers. Hence, ordering constraints can be checked in polynomial time. In [15], a subsumption decision algorithm was presented, running in exponential time in the number of literals. Furthermore, the subsumption check is performed at most for each pair of closures. Hence, subsumption checking takes exponential time in $|KB|$. Each closure can potentially participate in an inference with each other closure, resulting in c^2 combinations. Furthermore, an inference rule can be applied to any pair of literals, resulting in l^2 combinations. Finally, any of the 5 inference rules may be applied. Hence, the number of applications of inference rules of $\mathcal{BS}_{DL}$ is bounded by $5c^2l^2$, which is exponential in $|KB|$, for unary coding of numbers. Now it is obvious that, after at most an exponential number of steps, the set of closures will be saturated, and the procedure will terminate. Since $\mathcal{BS}_{DL}$ is sound and complete with eager application of redundancy elimination rules, the claim of the theorem follows. □

In the proof of Theorem 1, we assumed an exponential algorithm for checking subsumption. In practice, it is known that modern theorem provers spend up to 90% of their time in subsumption checking, so an algorithm with a better worst case complexity on $\mathcal{ALCHIQ}^-$-closures is useful in practice.

Lemma 5. *Approximate subsumption checks for $\mathcal{ALCHIQ}^-$-closures may be performed in polynomial time.*

Proof. In [15] it was shown that subsumption between closures having at most one variable can be checked in polynomial time. This algorithm can be easily extended to additionally check η-reducibility, so subsumption checking for closures of type 3, 4, 5, 6 and 9 can be performed polynomially. Furthermore, checking whether a closure of type 1 or 2 subsumes some other closure can be performed by matching the negative literal first, and then checking whether there is a matching positive literal, which can be performed in quadratic time. The problematic cases are closures of type 7 and 8.

Let $\mathcal{C} = C \cdot \rho$ be a closure of type 7, with C containing the variable x and n variables y_i, and let $\mathcal{D} = D \cdot \rho$ be a closure of type 7, with D containing the variable x' and m variables y'_j. Let us first assume that none of the closures contains constants. For a closure α, let α^{Δ} denote the multiset of terms assigned to variables y_i in the substitution of α, and α^x denote the sub-multiset of literals of α containing only the variable x. $\mathcal{C}$

subsumes $\mathcal{D}$ if there exists a substitution σ, such that $C\rho\sigma \subseteq D\rho$. Obviously, σ will contain a mapping $x \mapsto x'$, and assignments from y_i to y'_j. Let $\sigma' = \{x \mapsto x'\}$.

The set of equality literals in $\mathcal{C}$ can be represented as a graph, where vertices are marked with variables y_i, and there is an edge between each pair of vertices. For $\mathcal{C}$ to subsume $\mathcal{D}$, it is necessary to embed such a graph from $\mathcal{C}$ into a graph from $\mathcal{D}$. Furthermore, vertices y_i, to which a value has been assigned in ρ, must be matched with such vertices y'_j, such that $y_i\rho\sigma' = y'_j\rho$. Obviously, such an embedding exists if and only if $n \leq m$ and $\mathcal{C}^{\Delta}\sigma' \subseteq \mathcal{D}^{\Delta}$, which can be checked in polynomial time. Notice that terms in equality literals always occur at substitution positions, so η-reducibility is always satisfied. Additionally, for $\mathcal{C}$ to subsume $\mathcal{D}$, we need to check whether $\mathcal{C}^{x}$ subsumes $\mathcal{D}^{x'}$, whether $\mathcal{C}^{y_i}$ subsumes $\mathcal{D}^{y'_j}$ for some y_i and y'_j (since closures are symmetric with respect to y_i, checking some y_i with some y'_j is sufficient), and whether the number of literals $\neg R(x, y_i)$ in $\mathcal{C}$ is smaller than or equal to the number of literals $\neg R(x, y'_j)$ in $\mathcal{D}$. These checks can be done in polynomial time.

Checking subsumption between closures of type 8 can be done in exactly the same way. Finally, checking whether a closure of type 7 subsumes a closure of type 8 differs only in the fact that σ will contain the mapping $x \mapsto g(x')$, so we set $\sigma' = \{x \mapsto g(x')\}$.

Furthermore, these checks may be performed on $\Xi(KB)$ before saturation. Since no inference with a closure of type 7 or 8 may remove an equality literal from the closure, if a closure $\mathcal{C}$ of type 7 or 8 does not subsume a closure $\mathcal{D}$ of type 7 or 8, no closure $\mathcal{C}'$ derived from $\mathcal{C}$ may subsume a closure $\mathcal{D}'$ derived from $\mathcal{D}$. If both $\mathcal{C}'$ and $\mathcal{D}'$ are derived from the same closure $\mathcal{C}$, if we resolve literals from left to right and use the ordering constraint inheritance from Lemma 3, any substitution σ will contain only mappings of the form $x \mapsto x'$ and $y_i \mapsto y'_i$, so we can replace the subsumption check by a simply checking syntactic equality of literals containing corresponding variables.

For the case where $\mathcal{C}$ or $\mathcal{D}$ contain ground subclauses, we may separately check the subsumption of the ground and the non-ground subclauses. This check is approximate: it is possible that the graph of equality literals from $\mathcal{C}$ can be embedded into the graph of equality literals from $\mathcal{D}$ by using some equalities from the ground part of $\mathcal{D}$. Furthermore, we cannot use this algorithm for checking whether a closure of type 7 subsumes a closure of type 9. However, we expect both of these situations to occur rarely, and therefore expect that approximate checks will be sufficient in most cases. □

3.6 Discussion

We briefly comment on some important aspects of our decision procedure. Firstly, it is important to note that basic superposition is crucial to obtain the decision procedure, as it it is responsible for restricting the depth of functional terms in any proof. If the basicness restriction were not employed, then from premises $g(h(x)) \approx f(h(x))$ and $C(f(g(x)))$, it would be possible to perform superposition into the term $g(x)$ of the second premise and obtain $C(f(f(h(x))))$. Under basic superposition, this inference is not applicable, since the subterm $g(x)$ is always introduced by a previous unification step. Thus, superposition into $g(x)$ of a term $f(g(x))$ is prohibited, so it does not increase the term depth.

The second important aspect is the fact that superposition into closures of type 3 is redundant. If this were not the case, from $f(g(x)) \approx h(g(x))$ and $R(x, f(x))$, one

might derive $R(g(x), f(g(x)))$, which can be turned into $R^-(f(g(x)), g(x))$. Now this and closures of type 3 and 7 create closures of ever increasing depth. Whereas most existing resolution decision procedures employ subsumption to limit the clause length [25, 14], to our best knowledge, our decision procedure is the first one which uses subsumption to restrict the term depth.

In a way, basic superposition 'remembers' that $f(g(x))$ is actually a successor of $g(x)$ in a tree, so $g(x)$ is always marked. Because of that, each closure of type 6 contains exactly one such $g(x)$. Furthermore, superposition into closures of type 3 is redundant, since each closure 'remembers' that each functional term appearing in it was generated by some closure of type 3. These two features are closely related to the tree model property of the $\mathcal{ALCHIQ}^-$ description logic.

We briefly comment on why the restrictions to very simple roles is necessary and why basic superposition as such does not decide the unrestricted $\mathcal{ALCHIQ}$ variant. In essence, the restriction to very simple roles in number restrictions is necessary for establishing properties 5.i, 6.v, 6.vi, 7.viii, 8.vii, 8.viii and 9.iv. These properties are used to show that superposition into a generator is redundant, since the conclusion is subsumed by another generator. If at-most restrictions are allowed on roles having subroles, then this subsuming generator need not exist. Consider the following knowledge base: $R \sqsubseteq T, S \sqsubseteq T, C \sqsubseteq \exists R.\top, \top \sqsubseteq \exists S^-.\top, \top \sqsubseteq \leq 1\, T$. The following is the translation into closures:

$$\neg C(x) \vee R(x, f(x)) \tag{1}$$
$$S^-(x, g(x)) \tag{2}$$
$$\neg R(x, y) \vee T(x, y) \tag{3}$$
$$\neg S(x, y) \vee T(x, y) \tag{4}$$
$$\neg S^-(x, y) \vee S(y, x) \tag{5}$$
$$\neg T(x, y_1) \vee \neg T(x, y_1) \vee y_1 \approx y_2 \tag{6}$$

In the following presentation, we omit the markers to reduce the clutter. From (2) and (5), one can derive $S(g(x), x)$. From this and from (1), (3), (4) and (6) it is possible to derive $f(g(x)) \approx x$, which can be superposed back into (1), resulting in a closure $C(g(x)) \vee R(g(x), x)$. However, since T was used in an at-least restriction and is not very simple, condition 6.vi is not satisfied for $f(g(x)) \approx x$: the designated role of f is R, but the inner occurrence of $g(x)$ stems from $\mathbf{P^g}(x) \vee S([g(x)], x)$. Hence, the superposition conclusion is not redundant, but it is also not an $\mathcal{ALCHIQ}^-$-closure. It is easy to construct examples where such closures yield terms of ever increasing depth.

We finish this discussion with a short note on the complexity of our algorithm. From [30] it is known that deciding satisfiability of $\mathcal{ALCHIQ}^-$ knowledge bases is ExpTime-complete, even for non-unary coding of number in the input. Our algorithm runs in exponential time only if unary coding of numbers in input is assumed. Practically, this means that the algorithm should not be used for large numbers in cardinality restrictions, as this may degrade the performance.

4 Reducing $\mathcal{ALCHIQ}^-$ to Disjunctive Datalog

Based on the decision procedure from Section 3, in this section we show how to reduce an $\mathcal{ALCHIQ}^-$ knowledge base KB to a disjunctive datalog program. Marking information is not relevant for the reduction to datalog, so in this section we consider any closure $C \cdot \sigma$ equivalent to the clause $C\sigma$.

4.1 Overview

The translation we present is based on the observation that all ground functional terms encountered during saturation step of algorithm $\mathsf{Consistent}(KB)$ are of depth at most one. Hence, each such ground functional term can be simulated by a fresh constant. Hence, one can simulate ground inference steps of $\mathcal{BS}_{DL}$ in a function-free version of KB. The algorithm proceeds as follows:

- The TBox and RBox clauses of KB are first saturated under $\mathcal{BS}_{DL}$. As shown by Lemma 6, certain clauses can be removed from the saturated set, as they may not participate in further inferences.
- If the saturated set does not contain the empty clause, function symbols are eliminated from saturated clauses cf. Subsection 4.2. Lemma 7 demonstrates that this transformation does not affect satisfiability.
- In order to reduce the size of the datalog program, some irrelevant rules may be removed, cf. Subsection 4.3. Lemma 8 demonstrates that this transformation also does not affect satisfiability.
- Transformation of KB into a disjunctive datalog program, cf. Subsection 4.4, is now straightforward: apart from some technical assumptions, it suffices to transform each clause into equivalent sequent form. This transformation does not affect entailment, which is trivially demonstrated by Theorem 2.

4.2 Eliminating Function Symbols

For an $\mathcal{ALCHIQ}^-$ knowledge base KB, let $\Gamma_{\mathcal{TR}} = \Xi(KB_{\mathcal{T}} \cup KB_{\mathcal{R}})$. Let $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}})$ denote the *relevant set of saturated clauses*, that is, clauses of type 1, 2, 5, 7 obtained by saturating $\Gamma_{\mathcal{TR}}$ using $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules. Finally, let $\Gamma = \mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}}) \cup \Xi(KB_{\mathcal{A}})$. Intuitively, $\mathsf{Sat}(\Gamma_{\mathcal{TR}})$ contains all non-redundant clauses following from the TBox and RBox. From this clause set any further inference involved in deriving the empty clause will involve an ABox clause, which cannot participate in an inference with a clause of type 3, 4, 6 or 8. Hence, we may safely delete these clauses and consider only the $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}})$ subset.

Lemma 6. *KB is unsatisfiable if and only if Γ is unsatisfiable.*

Proof. KB is unsatisfiable if and only if the set of clauses derived by the saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$ contains the empty clause. Since choosing the premises of each inference rule is don't-care non-deterministic, we may perform all non-redundant inferences among clauses from $\Gamma_{\mathcal{TR}}$ first. Let us denote the resulting set of intermediate

clauses with $N_i = \mathsf{Sat}(\Gamma_{\mathcal{TR}}) \cup \Xi(KB_{\mathcal{A}})$. If N_i contains the empty clause, Γ contains it by definition as well (the empty clause is of type 5), and the claim of the Lemma follows. Otherwise, we continue with saturation of N_i. Obviously, each N_j, $j > i$, in the derivation, will be obtained from N_{j-1} by applying an inference rule involving at least one clause not in N_i, which can only be a clause of type 7 where u is a constant, or a clause of type 9. By Lemma 3, we may safely consider only derivations where the variables y_k in a clause are assigned terms in the decreasing order. Hence, N_j may not be obtained by resolving a clause of type 7 where u is a constant with a clause of type 3: this would assign y_k to $f(u)$, which is obviously larger than the constant that was assigned to some $y_{k'}$, $k' < k$. Furthermore, from the proof of Lemma 2, one may see that a clause of type 7 where u is a constant cannot participate in a resolution with a clause of type 3, since the unifier never exists. The same lemma shows that any other inferences with clauses of types 3, 4, 6 or 8 are either not possible, or are redundant. Therefore, we may conclude that no clause of type 3, 4, 6 or 8 from N_i participates in deriving N_j, $j > i$. Hence, N_i may safely be replaced by Γ. Any set of clauses N_j, $j > i$, which can be obtained by saturation from $\Xi(KB)$ may be obtained by saturation from Γ as well, modulo clauses of type 3, 4, 6 or 8. Hence, the saturation of Γ by $\mathcal{BS}_{DL}$ derives the empty clause if and only if the saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$ derives the empty clause, so the claim of the Lemma follows. □

We now show how to eliminate function symbols from clauses in Γ. Intuitively, the idea is to replace each ground functional term $f(a)$ with a new constant, denoted as a_f. For each function symbol f we introduce a new predicate symbol S_f, containing, for each constant a, a tuple of the form $S_f(a, a_f)$. Thus, S_f contains the f-successor of each constant. Any reference to a term $f(x)$ in some clause is then replaced with a new variable x_f, with the literal $\neg S_f(x, x_f)$ being added to the clause. Thus, for some a, resolving $\neg S_f(x, x_f)$ with $S(a, a_f)$ will bind the value of x_f to a_f, which plays the role of $f(a)$. The Herbrand universe of the clause set becomes thus finite, so it can be represented as a finite relation HU containing all constants a and a_f, and is used to bind unsafe variables.

In order to formalize this process, we first define an operator λ which eliminates functional terms and binds all unsafe variables in a clause.

Definition 7. *Let KB be an $\mathcal{ALCHIQ}^-$ knowledge base. For some ground functional term $f(a)$, let $\lambda(f(a))$ denote a globally unique constant a_f, not occurring in KB [6]. For an $\mathcal{ALCHIQ}^-$-clause C, we define $\lambda(C)$ as follows:*

1. *For each term of the form $f(x)$ in C, introduce a fresh variable x_f not occurring in C. Replace each occurrence of $f(x)$ with x_f.*
2. *Replace each ground functional term $f(a)$ with $\lambda(f(a))$.*
3. *For each variable x_f introduced in the first step, append the literal $\neg S_f(x, x_f)$.*
4. *If after steps 1–3 some variable x occurs in a positive literal but not in a negative literal, append the literal $\neg HU(x)$.*

If p is a position in a clause C, let $\lambda(p)$ denote the corresponding position in $\lambda(C)$. Let λ^- denote the inverse of λ (i.e. $\lambda(\lambda^-(C)) \equiv C$ for any clause C).

[6] Globally unique means that, for some f and a, the constant a_f is always the one and the same.

Let $\mathsf{FF}(KB) = \mathsf{FF}_\lambda(KB) \cup \mathsf{FF}_{Succ}(KB) \cup \mathsf{FF}_{HU}(KB) \cup \Xi(KB_\mathcal{A})$ *denote the function-free version of* $\Xi(KB)$*, where* FF_λ*,* FF_{Succ} *and* FF_{HU} *are defined as follows, where* a *and* f *range over all constant and function symbols in* $\Xi(KB)$*:*

$$\begin{aligned}\mathsf{FF}_\lambda(KB) &= \textstyle\bigcup_{C \in \mathsf{Sat}_\mathsf{R}(\Gamma_{\mathcal{TR}})} \lambda(C)\\ \mathsf{FF}_{Succ}(KB) &= \textstyle\bigcup S_f(a, \lambda(f(a)))\\ \mathsf{FF}_{HU}(KB) &= \textstyle\bigcup HU(a) \cup \bigcup HU(\lambda(f(a)))\end{aligned}$$

We now show that KB and $\mathsf{FF}(KB)$ are equi-satisfiable.

Lemma 7. *KB is unsatisfiable if and only if $\mathsf{FF}(KB)$ is unsatisfiable.*

Proof. We show that Γ and $\mathsf{FF}(KB)$ are equi-satisfiable. Since KB and Γ are equi-satisfiable by Lemma 6, the claim of the lemma follows.

($\Leftarrow$) If $\mathsf{FF}(KB)$ is unsatisfiable, since hyperresolution with superposition and splitting is sound and complete [3], a derivation of an empty clause exists. We now show that each such a derivation can be reduced to a derivation of the empty clause in Γ by sound inference rules, in particular, hyperresolution, paramodulation, instantiation and splitting. In $\mathsf{FF}(KB)$, all clauses are safe, so electrons are always positive ground clauses, and each hyperresolvent is a positive ground clause. Furthermore, since superposition into variables is not necessary for completeness, superposition-related inferences are necessary only among ground clauses. Finally, splitting ground clauses simplifies the proof, since all ground clauses on each branch are unit clauses.

Let B be a branch branch $\mathsf{FF}(KB) = N_0, \ldots, N_n$ of a derivation by hyperresolution with superposition and eager splitting from $\mathsf{FF}(KB)$. We show now by induction on n that, for any branch B, there exists a corresponding branch B' in a derivation from Γ by sound inference steps, and a set of clauses N'_m on B' such that: (*) if C is some clause in N_n not of the form $S_f(u, v)$ or $HU(u)$, then N'_m contains the *counterpart clause* of C, equal to $\lambda^-(C)$. The induction base $n = 0$ is obvious, as $\mathsf{FF}(KB)$ and Γ contain only one branch, on which, other than $S_f(u, v)$ or $HU(u)$, all ground clauses are ABox clauses. Now assume that the proposition (*) holds for some n and consider all possibilities for the inference of a clause C from clauses in N_n, forming N_{n+1}:

- Superposition into a literal $HU(u)$ is redundant, since HU is instantiated for each constant occurring in $\mathsf{FF}(KB)$, so the conclusion already appears on the branch.
- Assume that the inference is a superposition from $s \approx t$ into the ground unit clause L. If L is of the form $S_f(u, v)$, then the proposition obviously holds. Otherwise, clauses $s \approx t$ and L are derived in at most n steps on B, so by induction assumption counterpart clauses $\lambda^-(s \approx t)$ and $\lambda^-(L)$ are derivable in B'. Thus, superposition can be performed on these clauses in B', so the proposition holds.
- Reflexivity resolution can only be performed on some clause $u \not\approx u$ in B. By induction hypothesis $\lambda^-(u \not\approx u)$ is then derivable in B', and reflexivity resolution can be applied there, so the proposition holds.
- Equality factoring is not applicable to B, since all positive clauses in B are ground unit clauses.

– Assume that the inference is a hyperresolution inference with nucleus C, the set of positive ground electrons $E_1, \ldots, E_k$, and the unifier σ, resulting in the hyperresolvent H. We construct the substitution σ' as follows: for each variable $x \in \mathsf{dom}(\sigma)$ not of the form x_f, we include a mapping $x \mapsto \lambda^-(x\sigma)$. Let us now perform on B' an instantiation step $C' = (\lambda^-(C))\sigma'$. Obviously, $\lambda^-(C\sigma)$ and C' may differ only at a position p in C, at which a variable of the form x_f occurs. Let us denote with p' the position $\lambda^-(p)$ in C'. Furthermore, the term at p' in $\lambda^-(C)$ is $f(x)$, so with p'_x we denote the position of the inner x in $f(x)$. In the hyperresolution inference generating H, the variable x_f is instantiated by resolving $\neg S_f(x, x_f)$ with some ground literal $S_f(u, v)$. Hence, $C\sigma$ contains at p the term v, whereas C' contains at p' the term $f(u)$, and $\lambda^-(v) \neq f(u)$. We show now how to eliminate all such discrepancies in B'. Observe that the literal $S_f(u, v)$ is on B obtained from some $R = S_f(a, a_f)$ by n or less superposition inference steps. Let us denote by Δ_1 (Δ_2) the sequence of ground unit equalities applied to the first (second) argument of R. All $s_i \approx t_i$ from Δ_1 or Δ_2 are derivable in n steps or less on B, so corresponding equalities $\lambda^-(s_i \approx t_i)$ are derivable on B' by induction hypothesis. Let us denote these corresponding sequences with Δ'_1 and Δ'_2. We may now perform superposition with equalities from Δ'_1 to C' at p'_x in the reverse order. After this, p'_x will contain the constant a, and p' will contain the term $f(a)$. Hence, we may now apply superposition with equalities from Δ'_2 at p' in the original order. After this is done, each position p' will contain the term $\lambda^-(v)$. Let us denote with C'' the result of removing discrepancies at all positions. Obviously, $C'' = \lambda^-(C\sigma)$. All electrons E_i are derivable in n steps or less on B, so if E_i is not of the form $S_f(u, v)$ or $HU(u)$, $\lambda^-(E_i)$ is derivable on B'. We may now hyperresolve these electrons with C'' to obtain H'. Obviously, $H' = \lambda^-(H)$, so the proposition holds.
– If some ground clause C of length k causes the branch B to be split into k sub-branches, then $\lambda^-(C)$ is also of length k and B' can be split into k sub-branches, where each of them satisfies (*), so the proposition holds.

Hence, if there is a derivation of the empty clause on all branches from $\mathsf{FF}(KB)$, then there is a derivation of the empty clause on all branches from Γ as well.

($\Rightarrow$) If Γ is unsatisfiable, since $\mathcal{BS}_{DL}$ is sound and complete, a derivation of an empty clause exists. We now show that each such derivation can be reduced to a derivation of the empty clause in $\mathsf{FF}(KB)$ by sound inference rules.

Let B' be a derivation $\Gamma = N'_0, \ldots, N'_n$ by $\mathcal{BS}_{DL}$. We show by induction on n that there exists a corresponding derivation B of the form $\mathsf{FF}(KB) = N_0, \ldots, N_m$ by sound inference steps, such that: (**) if C' is some clause in N'_n, then N_m contains the counterpart clause $C = \lambda(C')$. The induction base $n = 0$ is trivial. Assume now that (**) holds for some n and consider possible inferences deriving $N'_{n+1} = N'_n \cup \{C'\}$, where the clause C' is derived from premises P'_1 and P'_2 in N'_n. By induction hypothesis, we know that there is a derivation B from $\mathsf{FF}(KB)$ with a clause set N_m containing the counterpart clauses of the premises P'_1 and P'_2, denoted with P_1 and P_2, respectively. We now consider each possible inference that might have lead to the derivation of C' and show how to construct a derivation of $C = \lambda(C')$ from N_m.

Assume that the inference is by ordered resolution on literals $L'_1 \in P'_1$ and $L'_2 \in P'_2$. Then resolution may be applied on corresponding literals L_1 and L_2 of P_1 and P_2, respectively, resulting in a clause D. Unification of a non-ground functional term $f(x)$ with some other term or variable in L'_1 and L'_2 corresponds to the unification of x_f with some other term or variable in L_1 and L_2. The differences between $\lambda(C')$ and D may have the following causes:

- C' may have some term $f(a)$ appearing in C' at position p, while D contains x_f at $\lambda(p)$. However, D then contains the literal $\neg S_f(a, x_f)$, which can be resolved with $S(a, a_f)$, to produce a_f at position $\lambda(p)$.
- $\lambda(C')$ and D may differ in some literal of the form $\neg HU(u)$. Since, for any constant u, any set of clauses on B contains $HU(u)$, this discrepancy can easily be removed by resolving C with $HU(u)$.

By successively removing differences between D and $\lambda(C')$, we eventually obtain a clause C such that $C = \lambda(C')$.

If the inference is by equality factoring or reflexivity resolution, then the premise P'_1 is ground and the inference may be applied to P_1 in the same way.

Assume the inference is by positive or negative basic superposition. If both P_1 and P_2 are ground, since superposition into Skolem function symbols is not needed, superposition can be applied to P_1 and P_2 in the same way. Otherwise, P_1 is a clause of type 5. Let superposition be performed at position p into a term of the form $f(x)$ with the term in P_2 being of the form $f(a)$, with unifier $\{x \mapsto a\}$. This inference can be simulated in N_m as follows: P_1 must contain a literal $\neg S_f(x, x_f)$ and the variable x_f must occur at position $\lambda(p)$. One can first resolve P_1 with $S_f(a, a_f)$, which will produce a_f at position $\lambda(p)$. Now one may perform superposition with P_2 at $\lambda(p)$ to obtain the clause C. Since P_1 contains x_f, is it safe and does not contain any $\neg HU(x)$ literals, so $C = \lambda(C')$.

Hence, if there is a derivation of the empty clause from Γ, then there is a derivation of the empty clause from $\mathsf{FF}(KB)$ as well. □

The result above means that $KB \models \alpha$ if and only if $\mathsf{FF}(KB) \models \alpha$, where α may be of the form $(\neg)A(a)$ or $(\neg)R(a, b)$, where A is an atomic concept. The proof also reveals the fact that, in checking satisfiability of $\mathsf{FF}(KB)$, it is not necessary to perform superposition into literals $HU(a)$.

In case the knowledge base uses only constructs from the $\mathcal{ALCHI}$ subset, further optimizations are possible, since $\Xi(KB)$ then does not contain equalities. The proof of Lemma 2 implies that clauses of type 5 containing a functional term cannot participate in any inference with clauses of type 9: superposition into $f(x)$ is not possible, so no ground literal containing a functional term may be generated. In this case, $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}})$ should contain only function-free clauses from the saturated set. Also, $\mathsf{FF}(KB)$ should contain only $HU(a)$ for each constant a.

4.3 Removing Irrelevant Clauses

The saturation of $\Gamma_{\mathcal{TR}}$ derives new clauses which enable the reduction to $\mathsf{FF}(KB)$. However, the same process introduces lots of clauses which are not necessary. Consider,

for example, the knowledge base $KB = \{A \sqsubseteq C, C \sqsubseteq B\}$. If the predicate ordering is $C \succ B \succ A$, then the saturation process will derive the clause $\neg A(x) \vee B(x)$, which is not necessary: all ground consequences of this clause may be obtained by combining ground consequences of the first two. Hence, in this subsection we present an optimization, by which we reduce the number of clauses in the resulting disjunctive datalog program.

Definition 8. *Let $C \in \mathsf{FF}(KB)$ be a clause such that $\lambda^-(C)$ was derived in the saturation of $\Gamma_{\mathcal{TR}}$ from premises P_i, $1 \leq i \leq k$, by an inference with a substitution σ. Then C is* irrelevant *in $\mathsf{FF}(KB)$ if, for each premise P_i, $\lambda(P_i)$ is defined, $\lambda(P_i) \in \mathsf{FF}(KB)$, and each variable occurring in $\lambda(P_i\sigma)$ occurs in C. A clause C is* relevant *if and only if it is not irrelevant. Finally, we use $\mathsf{FF}_\mathsf{R}(KB)$ to set of all clauses relevant in $\mathsf{FF}(KB)$.*

Removing irrelevant clauses preserves satisfiability, as demonstrated by the following lemma.

Lemma 8. *$\mathsf{FF}_\mathsf{R}(KB)$ is unsatisfiable if and only if $\mathsf{FF}(KB)$ is unsatisfiable.*

Proof. Let C be an irrelevant clause in $\mathsf{FF}(KB)$, where $\lambda^-(C)$ is derived in the saturation of $\Gamma_{\mathcal{TR}}$ from premises P_i by an inference rule ξ with a substitution σ. Let N be a (not necessarily proper) subset of $\mathsf{FF}(KB)$, such that $C \in N$ and $\lambda(P_i) \in N$, $i \leq i \leq k$. We now demonstrate the following property (***): N is unsatisfiable if and only if $N \setminus \{C\}$ is unsatisfiable. The ($\Leftarrow$) direction is trivial, since $N \setminus \{C\} \subset N$.

For the ($\Rightarrow$) direction, by Herbrand's theorem, N is unsatisfiable if and only if some finite set M of ground instances of N is unsatisfiable. For such M, we construct the set of ground clauses M' in the following way, where $\lambda(\sigma)$ is the substitution obtained from σ by changing each $x \mapsto t$ into $x \mapsto \lambda(t)$:

- For each $D \in M$ such that D is not a ground instance of C, let $D \in M'$.
- For each $D \in M$ such that D is a ground instance of C with substitution τ, let $\lambda(P_i)\lambda(\sigma)\tau \in M'$, $1 \leq i \leq k$.

Let τ be a ground substitution for C and $D = C\tau$. Since P_i can be clauses of type 1 – 5, and σ is the most general unifier, it can contain only mappings of the form $x \mapsto c$, $x \mapsto x'$ or $x \mapsto f(x')$. Hence, the set of variables in $\lambda(P_i\sigma)$ and $\lambda(P_i)\lambda(\sigma)$ coincide, and since τ instantiates all variables from $\lambda(P_i\sigma)$, the clauses in M' are indeed ground instances of $N \setminus \{C\}$. Furthermore, it is easy to see that $\lambda(P_i)\lambda(\sigma)\tau \subseteq \lambda(P_i\sigma)\tau$. If the inclusion is strict, this is due to literals of the form $\neg S_f(a, b)$ in the latter clause which are not in the first one because σ instantiates some variable from P_i to a functional term $f(x')$ originating from some premise P_j. But then $\lambda(P_j)$ contains the literal $\neg S_f(x', x'_f)$, so $\lambda(P_j)\lambda(\sigma)\tau$ contains $\neg S_f(a, b)$. Therefore, all $\lambda(P_i)\lambda(\sigma)\tau$ can participate in an ground inference corresponding to ξ deriving D, so if M is unsatisfiable, M' is unsatisfiable as well. Since M' is an unsatisfiable set of ground instances of $N \setminus \{C\}$, $N \setminus \{C\}$ is unsatisfiable by Herbrand's theorem.

Let *derives* be a binary relation on clauses in $\mathsf{FF}(KB)$, such that C_1 *derives* C_2 if $\lambda^-(C_1)$ was used as a premise for deriving $\lambda^-(C_2)$ in the saturation of $\Gamma_{\mathcal{TR}}$. Obviously, *derives* is a directed acyclic graph, so it can be topologically sorted into a sequence

$C_1, \ldots, C_n$, such that for each $1 \leq i < j \leq n$, no C_i *derives* some C_j (i.e. each clause has a smaller index than the clauses it was derived from). Consider now a sequence of clause sets $N_0 = \mathsf{FF}_\mathsf{R}(KB), N_1, \ldots, N_n$, where $N_i = N_{i-1}$ if C_i is relevant in $\mathsf{FF}(KB)$, and $N_i = N_{i-1} \setminus \{C_i\}$ if C_i is irrelevant in $\mathsf{FF}(KB)$, $1 \leq i \leq n$. By induction on n, N_n is unsatisfiable if and only if $\mathsf{FF}(KB)$ is unsatisfiable: if C_i is irrelevant, since all premises deriving $\lambda^-(C_i)$ are in N_i, the conditions of (***) are satisfied. Furthermore, all irrelevant clauses are eliminated in N_n. Hence, $N_n = \mathsf{FF}_\mathsf{R}(KB)$, and the claim of the lemma follows. □

4.4 Reduction to Disjunctive Datalog

Reduction of an $\mathcal{ALCHIQ}^-$ knowledge base KB to disjunctive datalog is now easy.

Definition 9. *Reduction of KB to a disjunctive datalog program $\mathsf{DD}(KB)$ is obtained by simply rewriting each clause $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$ in $\mathsf{FF}_\mathsf{R}(KB)$ as the rule $A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$.*

Theorem 2. *Let KB be an $\mathcal{ALCHIQ}^-$ knowledge base and let $\mathsf{DD}(KB)$ be its reduction to disjunctive datalog. Then the following claims hold:*

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

Proof. The first claim is an obvious consequence of Lemma 8. The second claim follows from the first one, since $\mathsf{DD}(KB \cup \{\neg\alpha\}) = \mathsf{DD}(KB) \cup \{\neg\alpha\}$ is unsatisfiable if and only if $\mathsf{DD}(KB) \models_c \alpha$. Furthermore, $KB \models C(a)$ if and only if $KB \cup \neg C(a)$ is unsatisfiable, which is the case if and only if $KB \cup \{\neg Q(a), \neg Q \sqsubseteq \neg C\} = KB \cup \{\neg Q(a), C \sqsubseteq Q\}$ is unsatisfiable. Now the third claim follows from the second one, and the fact that Q is atomic.

By Lemma 4, $|\mathsf{Sat}(\Gamma_{\mathcal{TR}})|$ is at most exponential in $|KB|$, and, for each clause C in it, the number of literals is at most polynomial in $|KB|$. It is easy to see that the application of λ to C can be performed in time polynomial in the number of terms and literals in C. The number of constants a_f added to $\mathsf{DD}(KB)$ is equal to $i \cdot f$, where i is the number of individuals, and f the number of function symbols. By Lemma 4, if numbers are unary coded, both i and f are polynomial in $|KB|$, so the number of constants a_f is also polynomial in $|KB|$. By Theorem 1, $\mathsf{Sat}(\Gamma_{\mathcal{TR}})$ can be computed in time at most exponential in $|KB|$, so the fourth claim follows. □

4.5 Answering Queries in $\mathsf{DD}(KB)$

In order to make this paper self-contained, we discuss briefly the techniques that can be used for query answering in $\mathsf{DD}(KB)$. Many techniques have been developed for disjunctive datalog without equality. These techniques can be used, provided that the usual congruence properties of equality are axiomatized correctly. This can be done by adding the following axioms to $\mathsf{DD}(KB)$, where the last axiom is instantiated for each predicate occurring in $\mathsf{DD}(KB)$ other than HU:

$$x \approx x \leftarrow HU(x). \tag{7}$$

$$x \approx y \leftarrow y \approx x. \tag{8}$$

$$x \approx z \leftarrow x \approx y, y \approx z. \tag{9}$$

$$P(\ldots, y, \ldots) \leftarrow P(\ldots, x, \ldots), x \approx y. \tag{10}$$

Currently, the state-of-the-art technique for reasoning in disjunctive datalog is so-called intelligent grounding [12]. The algorithm is based on model building, which is performed by generating the ground instantiation of the program rules, generating candidate models, and using model checking algorithms to eliminate models which do not satisfy the ground rules. In order to avoid generating the entire grounding of the program, carefully designed heuristics is applied to generate the subset of the ground rules which have exactly the same set of the stable models as the original program. Query answering is reduced to model building, since A is not a certain answer if and only if there is a model not containing A.

The intelligent grounding technique is currently the state-of-the-art technique for reasoning in disjunctive datalog and has been implemented successfully in the DLV disjunctive datalog engine [12]. However, it is important to understand that in disjunctive datalog applications, computing the models is usually of more interest than query answering. For example, disjunctive datalog has been successfully applied to planning problems, where each model usually represents one computed plan.

On the contrary, the models of $\mathsf{DD}(KB)$ are of virtually no interest and $\mathsf{DD}(KB)$ does not contain negation-as-failure. Hence, we propose query answering in $\mathsf{DD}(KB)$ by hyperresolution and basic superposition, which may be viewed as an extension of the fixpoint computation of plain datalog. A similar technique was presented in [9]. However, the algorithm presented there has two drawbacks: it does not take equality into account, and it does not specify whether application of redundancy elimination techniques is allowed. Roughly, the technique consists of saturating the rules and facts of $\mathsf{DD}(KB)$ by hyperresolution with basic superposition under an ordering in which all ground query literals are smallest. Additionally, it is required that the query predicate does not occur in the body of any rule. Under these assumptions, one may show that the saturated set of clauses will contain all ground query literals which are cautiously entailed by the program. Because the ordering is total, in each ground disjunction there is exactly one maximal literal. Hence, semi-naive bottom-up computation or join order optimizations can be adapted to the disjunctive case.

It is important to understand that even though we do not propose reusing intelligent grounding, the reduction to disjunctive datalog has the benefit that it enables the application of the magic sets transformation [16]. This technique is independent of the actual query evaluation strategy and can be reused as-is. We find it difficult to adapt the technique to other formalisms.

We now present the query answering technique in more detail. Let $\mathcal{HBS}$ denote the $\mathcal{BS}$ calculus where ordered resolution rule is replaced with hyperresolution of all negative literals. Let P be a positive satisfiable disjunctive datalog program and Q a predicate not occurring in the body of any rule in P. Let $\mathsf{Sat}_{\mathsf{HBS}}(P)$ denote the set of ground closures obtained by saturating P by $\mathcal{HBS}$ under any term ordering in which all literals of the form $Q(\mathbf{a})$ are smallest. Furthermore, apart from usual basic superposition inferences, for any two closures $s \approx t \vee C$ where $s \approx t$ is strictly maximal and $s \succ t$, and $Q(\mathbf{a}) \vee D$ where $Q(\mathbf{a})$ is strictly maximal, we perform any possible superposition from t into $Q(\mathbf{a})$, and we perform it even if the corresponding position in $Q(\mathbf{a})$ is marked. During saturation, redundancy elimination techniques may be applied as usual.

A remark about the above definition is in order. Namely, any admissible ordering is stable under contexts and under substitutions, and is total on ground terms; therefore, it has the subterm property for ground terms [2]. However, in such an ordering a literal $Q(\mathbf{a})$ is not always the smallest: for literals $a \approx b$ and $Q(a)$ with $a \succ b$, we always have $Q(a) \succ a$, so $Q(a) \succ a \approx b$.

This situation can be remedied by dropping the requirement on $\succ$ to be stable under substitutions (i.e. that for all substitutions σ and terms s and t, $s \succ t$ implies $s\sigma \succ t\sigma$). Hence, for ground terms the term ordering $\succ$ must be total, well-founded and stable under contexts (i.e. for all ground terms s, t and u, and all positions p, $s \succ t$ implies $u[s]_p \succ u[t]_p$). An example is a *query* ordering $\succ_Q$ induced over a total precedence of over constant symbols $>_C$ and predicate symbols $>_P$ such that Q is the smallest element in $>_Q$, defined in the following way ($a_{(i)}$ and $b_{(i)}$ are arbitrary constants, P and R are arbitrary predicate symbols, and Q is the query predicate symbol):

- $a \succ_Q b$ if $a >_C b$,
- $P(a_1, \ldots a_n) \succ_Q b$,
- $P(a_1, \ldots a_n) \succ_Q R(b_1, \ldots, b_m)$ if $R >_P R$,
- $P(a_1, \ldots a_n) \succ_Q P(b_1, \ldots, b_n)$ if there is some k, $1 \leq k \leq n$, such that $a_i = b_i$ for $i < k$ and $a_k \succ b_k$,
- $a \succ_Q Q(b_1, \ldots, b_n)$.

It is easy to see that $\succ_Q$ is well-founded, stable under contexts, and that it fulfills the requirements of $\mathcal{HBS}$. However, it is not defined for non-ground terms, since extending $\succ_Q$ to non-ground terms would require $x \succ Q(x)$. Therefore, $\succ_Q$ cannot be used to decide satisfiability of a general first-order theory by $\mathcal{BS}$.

However, satisfiability of a positive program P can be decided by saturating P under $\mathcal{HBS}$. Let P_∞ be the set of closures obtained by saturating P under $\mathcal{HBS}$ and consider applying model-generation method to P_∞. All non-ground closures in P_∞ have selected negative literals, so they are not productive. Therefore, all productive clauses in P_∞ are positive ground clauses without functional terms. Literals from such clauses can be compared using $\succ_Q$, and a model can be generated in the same way as

in [7, 24]. In fact, to saturate P and to generate a model of P_∞ it is not necessary to compare non-ground terms, so stability under substitutions is not needed. Therefore, we conclude that $\mathcal{HBS}$ is sound and complete for deciding satisfiability of P. From this we obtain the following result:

Lemma 9. *$P \models_c Q(\mathbf{a})$ if and only if $Q(\mathbf{a}) \in \mathsf{Sat}_{\mathsf{HBS}}(P)$.*

Proof. $P \models_c Q(\mathbf{a})$ if and only if the set of closures N obtained by saturating $P \cup \{\neg Q(\mathbf{a})\}$ under hyperresolution and basic superposition under the above specified ordering contains the empty closure. Notice that, since all rules in P are safe, all hyperresolvents are positive ground closures.

Consider first the case when no superposition inference is applied to the literal $\neg Q(\mathbf{a})$ in the saturation of N. Since P is satisfiable, N contains the empty closure if and only if a hyperresolution with $\neg Q(\mathbf{a})$ is performed in saturation. Since the literals containing Q are smallest in the ordering, a positive literal $Q(\mathbf{a})$ can be maximal only in a closure $C = Q(\mathbf{a}) \vee D$, where D contains only literals with the Q predicate. Since $\neg Q(\mathbf{a})$ is the only closure where Q occurs negatively, if D is not empty, no literal from D can be eliminated by a subsequent hyperresolution inference. Hence, the empty closure can be derived from such C if and only if D is empty, which is the case if and only if $Q(\mathbf{a}) \in \mathsf{Sat}_{\mathsf{HBS}}(P)$.

Assume now that in the saturation of N several superposition inferences from closures $a_i \approx b_i \vee C_i$, $a_i \succ b_i$, are applied to $\neg Q(\mathbf{a})$, resulting in a closure $\neg Q(\mathbf{b}) \vee C$, which then participates in a resolution with a closure $Q(\mathbf{b}) \vee D$ yielding $C \vee D$. Such a derivation can be transformed into a derivation where superposition inferences are performed on b_i into $Q(\mathbf{b}) \vee D$, yielding $Q(\mathbf{a}) \vee C \vee D$, which can then participate in a resolution with $\neg Q(\mathbf{a})$ to obtain $C \vee D$. Thus, we may successively eliminate each superposition into some $\neg Q(\mathbf{a})$ and obtain a derivation in which no superposition into some $\neg Q(\mathbf{a})$ has been performed. Since in $\mathsf{Sat}_{\mathsf{HBS}}(P)$ all superposition inferences from the smaller side of the equality are performed into all literals containing Q, and all such inferences are sound, the claim of the lemma follows. □

Assuming that Q is a single predicate, or that it does not occur in the body of any rule in P does not reduce the generality of the approach, as one can always add a new rule of the form $Q(\mathbf{x}) \leftarrow \mathbf{A}(\mathbf{x})$ so that the conditions of Lemma 9 are satisfied.

We now consider the complexity of query answering in $\mathsf{DD}(KB)$. Namely, it is well-known that the expression complexity of checking whether $P \models_c A$ for any program P is co-NEXPTIME$^{\mathrm{NP}}$-complete in the size of $|P|$ [11]. Since $|\mathsf{DD}(KB)|$ is exponential in $|KB|$, a straight-forward reasoning gives an algorithm in co-2NEXPTIME$^{\mathrm{NP}}$. However, in $\mathsf{DD}(KB)$, each rule is of length polynomial in $|KB|$, so the hardness arguments from [11] do not apply directly to $\mathsf{DD}(KB)$, since it is a program of a restricted form.

Theorem 3. *Computing the set of all ground literals of the form $C(a)$ or $R(a, b)$ which are entailed by $\mathsf{DD}(KB)$ can be done in time exponential in $|KB|$ for unary coding of numbers.*

Proof. In a way similar to the Lemma 4, it is easy to see that the maximal length of each ground clause in $\mathsf{Sat}_{\mathsf{HBS}}(\mathsf{DD}(KB))$ is polynomial in $|KB|$, so the number of ground clauses is exponential in $|KB|$. Furthermore, in each application of the hyperresolution inference to some rule r, one selects a clause for each body literal of r, which is polynomial in $|KB|$, giving rise to exponentially many different hyperresolution inferences. Hence, the saturation of $\mathsf{DD}(KB)$ may be performed in time exponential in $|KB|$. Since by Lemma 9 saturation of $\mathsf{DD}(KB)$ computes all certain answers of $\mathsf{DD}(KB)$, the claim of the theorem follows. □

We finish with a note about an optimization which may be applied if unique names assumption holds in KB and only answers involving the individual constants are of interest (which is the usual case). If one takes an ordering where all constants a_f are bigger than all individual constants a, then superposition inferences from the smaller side of an equation are not needed. Namely, since $a_f \succ a$, in saturation of $\mathsf{DD}(KB) \cup \{\neg Q(\mathbf{a})\}$ no superposition inference may replace some a_i with a_f. Furthermore, for any clause $D = a \approx b \vee C$, there is a clause $a \not\approx b$, so D may be replaced immediately with C. Hence, no equality can reduce a position in $\neg Q(\mathbf{a})$, so we always have the first case of the Lemma 9.

4.6 Discussion

It is important to note that our reduction to disjunctive datalog preserves entailment under *descriptive semantics*. Namely, in [23] Nebel has shown that knowledge bases containing terminological cycles are not definitorial. This means that, for some fixed partial interpretation of atomic concepts, several interpretations of non-atomic concepts may exist. It might be reasonable then to designate a particular interpretation as the intended one, with least and greatest fixpoint models being the obvious candidates. However, there is no definite answer to this question, as choosing either of the fixpoint models has its drawbacks. Consequently, most description logic systems implement the so called descriptive semantics, which coincides with that of Definition 2.

Obviously, our decision procedure implements exactly the descriptive semantics. Furthermore, Theorem 2 shows that $\mathsf{DD}(KB)$ entails exactly those ground facts which are entailed by our decision procedure, so $\mathsf{DD}(KB)$ also implements the descriptive semantics. Hence, one may say that the set of facts contained in any minimal model of $\mathsf{DD}(KB)$ coincides with the set of facts entailed by KB under descriptive semantics. Intuitively, it is the saturation process which is responsible for this.

Finally, we point out that basic superposition is crucial for the correctness of the translation. Namely, the basicness restriction renders superposition into Skolem function symbols redundant, which allows treating ground functional terms as constants.

5 Extension to $\mathcal{SHIQ}^-$ Description Logic

In this section we extend our results to $\mathcal{SHIQ}^-$, by encoding an $\mathcal{SHIQ}$ knowledge base into an equi-satisfiable $\mathcal{ALCHIQ}$ knowledge base. Restriction to very simple roles is not relevant in this context, so we prove our results for $\mathcal{SHIQ}$.

This transformation is similar to the one found in [30], where an algorithm for transforming $\mathcal{SHIQ}$ concepts to concepts in $\mathcal{ALCIQ}b$ logic was presented ($\mathcal{ALCIQ}b$ does not provide any role hierarchy, but allows certain types of boolean operations on roles). Another similar transformation has been presented in [29], where it is demonstrated, among others, how multi-modal logic with transitive modalities $\mathrm{K4}_m$ can be encoded in a multi-modal logic without transitive modalities K_m.

Definition 10. *For a $\mathcal{SHIQ}$ knowledge base KB, let $\mathsf{clos}(KB)$ denote the concept closure of KB, defined as the smallest set of concepts satisfying the following conditions:*

- *If $C \sqsubseteq D \in KB_{\mathcal{T}}$, then $\mathsf{NNF}(\neg C \sqcup D) \in \mathsf{clos}(KB)$.*
- *If $C \equiv D \in KB_{\mathcal{T}}$, then $\mathsf{NNF}(\neg C \sqcup D) \in \mathsf{clos}(KB)$ and $\mathsf{NNF}(\neg D \sqcup C) \in \mathsf{clos}(KB)$.*
- *If $C(a) \in KB_{\mathcal{A}}$, then $\mathsf{NNF}(C) \in \mathsf{clos}(KB)$.*
- *If $C \in \mathsf{clos}(KB)$ and D occurs in C, then $\{D, \mathsf{NNF}(\neg D)\} \subseteq \mathsf{clos}(KB)$.*
- *If $\forall R.C \in \mathsf{clos}(KB)$, $S \sqsubseteq^* R$, and $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$, then $\forall S.C \in \mathsf{clos}(KB)$.*

Notice that all concepts in $\mathsf{clos}(KB)$ are in NNF. Now we define the operator Ω which encodes any $\mathcal{SHIQ}$ knowledge base KB into an $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$ which can equivalently be used for satisfiability checking.

Definition 11. *For a $\mathcal{SHIQ}$ knowledge base KB, let $\Omega(KB)$ denote the following $\mathcal{ALCHIQ}$ knowledge base:*

- *$\Omega(KB)_{\mathcal{R}}$ is obtained from $KB_{\mathcal{R}}$ by removing all axioms $\mathsf{Trans}(R)$,*
- *$\Omega(KB)_{\mathcal{T}} = KB_{\mathcal{T}} \cup \{\forall R.C \sqsubseteq \forall S.(\forall S.C) \mid \forall R.C \in \mathsf{clos}(KB) \wedge S \sqsubseteq^* R \wedge \mathsf{Trans}(S) \in KB_{\mathcal{R}}\}$,*
- *$\Omega(KB)_{\mathcal{A}} = KB_{\mathcal{A}}$.*

Observe that, for any concept C, the number of subconcepts in $\mathsf{clos}(KB)$ is bounded by the number of subexpressions in C. Furthermore, for each concept from $\mathsf{clos}(KB)$, we may generate at most $|N_R|$ axioms in $\Omega(KB)_{\mathcal{T}}$. Hence, the encoding is polynomial in $|KB|$. We now show that this encoding does not affect satisfiability.

Theorem 4. *KB is satisfiable if and only if $\Omega(KB)$ is satisfiable.*

Proof. Out of convenience, we use the model-theoretic semantics of $\mathcal{SHIQ}$ in this proof. This semantics is identical to the one given by mapping to first-order logic. In particular, the notation $\alpha \in C^I$ should be understood as $I \models C(a)$, where $a^I = \alpha$.

($\Leftarrow$) Assume that I is a model of KB. We show that I is a model of $\Omega(KB)$ as well. The proof is by contradiction: assume that I is not a model of $\Omega(KB)$, so some axiom from $\Omega(KB)$ is not satisfied in I. Since we have $\Omega(KB)_{\mathcal{R}} \subseteq KB_{\mathcal{R}}$ and $KB_{\mathcal{T}} \subseteq \Omega(KB)_{\mathcal{T}}$, this axiom must have been added in the second point of Definition 11. Hence, there is a domain element α such that $\alpha \in \forall R.C^I$, but $\alpha \notin (\forall S.(\forall S.C))^I$. There are two possibilities:

- There is no domain element β for which $(\alpha, \beta) \in S^I$. Then $\alpha \in (\forall S.X)^I$, regardless of X. Hence, $\alpha \in (\forall S.(\forall S.C))^I$ must hold, which is a contradiction.

– There is β such that $(\alpha, \beta) \in S^I$. There are two possibilities:
  - If no domain element γ exists such that $(\beta, \gamma) \in S^I$, then $\beta \in (\forall S.(\forall S.C))^I$, which is a contradiction.
  - If there is γ such that $(\beta, \gamma) \in S^I$, by transitivity of S we have $(\alpha, \gamma) \in S^I$. Since and $S^I \subseteq R^I$ and $\alpha \in (\forall R.C)^I$, we have $\gamma \in C^I$. Since this holds for any γ, we have $\beta \in (\forall S.C)^I$. Since this holds for any β, we have that $\alpha \in (\forall S.(\forall S.C))^I$ and we derive a contradiction.

Since we derive a contradiction in all cases, I must be a model of $\Omega(KB)$.

($\Rightarrow$) As explained in Subsection 2.1, without loss of generality we may focus only on knowledge bases with acyclic RBoxes. Let R^+ denote the transitive closure of some relation R. Let I be a model of $\Omega(KB)$, and let interpretation I' be constructed from I as follows:

– For each atomic concept $A \in \mathsf{clos}(KB)$ set $A^{I'} = A^I$.
– If $\mathsf{Trans}(R) \in KB_{\mathcal{R}}$, set $R^{I'} = (R^I)^+$.
– If $\mathsf{Trans}(R) \notin KB_{\mathcal{R}}$, set $R^{I'} = R^I \cup \bigcup_{S \sqsubseteq^* R, S \neq R} S^{I'}$.

Since we may assume that $KB_{\mathcal{R}}$ is acyclic, the above induction is well-defined. We will now show that I' satisfies every RBox axiom of KB. By construction I' satisfies all transitivity axioms in $KB_{\mathcal{R}}$. Furthermore, I' satisfies each inclusion axiom in $KB_{\mathcal{R}}$: If R is not transitive, this is obvious from the construction; otherwise, this follows from the fact that $A^+ \cup B^+ \subseteq (A \cup B)^+$ for any A and B. Furthermore, for any role R we have $R^I \subseteq R^{I'}$. Finally, if R is simple, then $R^{I'} = R^I$.

We now show that, for each $D \in \mathsf{clos}(KB)$, $D^I \subseteq D^{I'}$. We do this by induction on the structure of D. For the base case, when D is an atomic concept A or a negation of an atomic concept $\neg A$, the claim follows directly from the definition of I'. For the induction steps we examine each possible form D might have:

– $D = C_1 \sqcap C_2$. Assume for some α we have $\alpha \in (C_1 \sqcap C_2)^I$. Then $\alpha \in C_1^I$ and $\alpha \in C_2^I$. By induction hypothesis, $\alpha \in C_1^{I'}$ and $\alpha \in C_2^{I'}$. Hence, $\alpha \in (C_1 \sqcap C_2)^{I'}$.
– $D = C_1 \sqcup C_2$. Assume for some α we have $\alpha \in (C_1 \sqcup C_2)^I$. Then either $\alpha \in C_1^I$, so by induction hypothesis $\alpha \in C_1^{I'}$, or $\alpha \in C_2^I$, so by induction hypothesis $\alpha \in C_2^{I'}$. Either way, $\alpha \in (C_1 \sqcup C_2)^{I'}$.
– $D = \exists R.C$. Assume for some α we have $\alpha \in (\exists R.C)^I$. Then there is a β such that $(\alpha, \beta) \in R^I$ and $\beta \in C^I$, so by induction hypothesis $\beta \in C^{I'}$. Since $R^I \subseteq R^{I'}$, we have $(\alpha, \beta) \in R^{I'}$. Hence, $\alpha \in (\exists R.C)^{I'}$.
– $D = \forall R.C$. Assume that $\alpha \in (\forall R.C)^I$. If there is no β such that $(\alpha, \beta) \in R^{I'}$, then $\alpha \in (\forall R.C)^{I'}$. Otherwise, assume there is such β. There are two possibilities:
  - $(\alpha, \beta) \in R^I$. Then $\beta \in C^I$, so by induction hypothesis $\beta \in C^{I'}$.
  - $(\alpha, \beta) \notin R^I$. Then there must be a role $T \sqsubseteq^* R$ with $\mathsf{Trans}(T) \in KB_{\mathcal{R}}$, and a path $(\alpha, \gamma_1) \in T^I$, $(\gamma_1, \gamma_2) \in T^I$, …, $(\gamma_{n-1}, \beta) \in T^I$, $n > 1$. But then $\forall R.C \sqsubseteq \forall T.(\forall T.C) \in \Omega(KB)_{\mathcal{T}}$, so $\alpha \in (\forall T.(\forall T.C))^I$, so $\gamma_1 \in (\forall T.C)^I$. Furthermore, $\forall T.C \sqsubseteq \forall T.(\forall T.C) \in \Omega(KB)_{\mathcal{T}}$, so for each i we have $\gamma_i \in (\forall T.C)^I$. For $n-1$ we have $\beta \in C^I$, so by induction hypothesis $\beta \in C^{I'}$.

Since for any β, in both cases we have that $\beta \in C^{I'}$, we have $\alpha \in (\forall R.C)^{I'}$.

- $D = \geq n\, R.C$. Assume that $\alpha \in (\geq n\, R.C)^I$. Then there are at least n distinct domain elements β_i such that $(\alpha, \beta_i) \in R^I$ and $\beta_i \in C^I$, so by induction hypothesis $\beta_i \in C^{I'}$. Since $R^I \subseteq R^{I'}$, we have $\alpha \in (\geq n\, R.C)^{I'}$.
- $D = \leq n\, R.C$. Since R is simple, $R^I = R^{I'}$. Let $E = \mathsf{NNF}(\neg C)$. Assume now that $\alpha \in (\leq n\, R.C)^I$, but $\alpha \notin (\leq n\, R.C)^{I'}$. Then a β must exist such that $(\alpha, \beta) \in R^I$, $\beta \notin C^I$, $\beta \in C^{I'}$, $\beta \in E^I$ and $\beta \notin E^{I'}$. However, since $E \in \mathsf{clos}(KB)$, by induction hypothesis we have $\beta \in E^{I'}$, which is a contradiction. Hence, $\alpha \in (\leq n\, R.C)^{I'}$.

Now it is obvious that any ABox axiom of the form $C(a)$ from KB is satisfied in I': since I is a model of $\Omega(KB)$, we have $a^I \in C^I$, but since $C^I \subseteq C^{I'}$, we have $a^I \in C^{I'}$. Also, any ABox axiom of the form $R(a, b)$ from KB is satisfied in I': since I is a model of $\Omega(KB)$, we have $(a^I, b^I) \in R^I$, but since $R^I \subseteq R^{I'}$, we have $(a^I, b^I) \in R^{I'}$. Finally, any TBox axiom of the form $C \subseteq D$ from KB is satisfied in I': since I is a model of $\Omega(KB)$, we have that $\triangle^I \subseteq (\neg C \sqcup D)^I$, but since $(\neg C \sqcup D)^I \subseteq (\neg C \sqcup D)^{I'}$, we have $\triangle^I \subseteq (\neg C \sqcup D)^{I'}$. Similar argument may be made for any TBox axiom of the form $C \equiv D$. □

Notice that $\Omega(KB \cup \{(\neg)C(a)\}) = \Omega(KB) \cup \{(\neg)C(a)\}$, so $KB \models (\neg)C(a)$ iff $\Omega(KB) \models (\neg)C(a)$. However, the models of KB and $\Omega(KB)$ may differ in the interpretation of complex roles, so $\Omega(KB)$ can be used only to prove entailment of ground facts $(\neg)R(a, b)$ for a simple role R.

6 Example

In this section we present a simple $\mathcal{ALCHIQ}^-$ knowledge base and show how it can be reduced to disjunctive datalog. Let KB denote the following knowledge base:

$$\geq 2\, hasChild \sqsubseteq TaxCut \tag{11}$$

$$Man \sqcap Woman \sqsubseteq \bot \tag{12}$$

$$\exists hasDaughter.\top \sqsubseteq Woman \tag{13}$$

$$\exists hasChild.(\exists hasDaughter.\top)(Peter) \tag{14}$$

$$hasChild(Peter, Paul) \tag{15}$$

$$Man(Paul) \tag{16}$$

The intuitive interpretation is as follows: (11) states that if someone has two children, then he is eligible for a tax cut. (12) says that the set of man and women are disjoint. (13) says that if someone has a daughter, then she is a woman. (14) says that $Peter$ has a child which has some daughter. Finally, (15) states that $Peter$ has a child $Paul$ and (16) says that $Paul$ is a man. Notice that axiom (14) states a form of incomplete knowledge: one knows that $Peter$ has at least one child, but one does not know its name. This is not allowed in traditional database data models, which can manage only definite information.

This knowledge base entails $TaxCut(Peter)$, and here is why: The unnamed child of $Peter$ implied by (14) has a daughter, so it must be a woman by (13). Furthermore, this unnamed child must be some other child than $Paul$, since $Paul$ is a man. Hence, $Peter$ has at least two children, so he is eligible for a tax cut. Notice that if we had no evidence that the unnamed child is a woman, we would not be able to tell whether this unnamed child is different from $Paul$, so we would not be able to draw the conclusion $TaxCut(Peter)$.

Let us now show how $TaxCut(Peter)$ may be concluded by reducing the knowledge base to disjunctive datalog. First we may observe that (14) is not extensionally reduced, so we introduce a new atomic concept Q_1 and replace (14) with these axioms:

$$Q_1 \sqsubseteq \exists hasChild.(\exists hasDaughter.\top) \tag{17}$$
$$Q_1(Peter) \tag{18}$$

The next step is to compute $\Xi(KB)$. We do this by applying the structural transformation, where we introduce a new predicate Q_2 for the subexpression $\exists hasDaughter.\top$ of (17). We define the precedence relation $>_P$ for the LPO as follows:

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

Then (19) to (24) below present the closures of $\Xi(KB_{\mathcal{T}})$, where literals which may participate in inferences have been underlined (they are either selected or maximal):

$$TaxCut(x) \vee \underline{\neg hasChild(x, y_1)} \vee \underline{\neg hasChild(x, y_2)} \vee y_1 \approx y_2 \tag{19}$$
$$\neg Man(x) \vee \underline{\neg Woman(x)} \tag{20}$$
$$\underline{\neg hasDaughter(x, y)} \vee \underline{Woman(x)} \tag{21}$$
$$\neg Q_1(x) \vee \underline{hasChild(x, f(x))} \tag{22}$$
$$\neg Q_1(x) \vee \underline{Q_2(f(x))} \tag{23}$$
$$\neg Q_2(x) \vee \underline{hasDaughter(x, g(x))} \tag{24}$$

Now we apply the inference rules of the basic superposition calculus in order to saturate the TBox. One can resolve (19) and (22) to obtain (25), (21) and (24) to obtain (26), (26) and (20) to obtain (27), and (23) and (27) to obtain (28). After these inferences are performed, $\Xi(KB_{\mathcal{T}})$ is saturated and consists of the following clauses:

$$TaxCut(x) \vee \neg Q_1(x) \vee \underline{\neg hasChild(x, y_2)} \vee [f(x)] \approx y_2 \tag{25}$$
$$\neg Q_2(x) \vee \underline{Woman(x)} \tag{26}$$
$$\underline{\neg Q_2(x)} \vee \neg Man(x) \tag{27}$$
$$\neg Q_1(x) \vee \underline{\neg Man(f(x))} \tag{28}$$

Closures (22) and (24) are of type 3 and are therefore not converted to disjunctive datalog. Furthermore, closures (27) and (28) are irrelevant, since they are obtained by ordered resolution into a negative literal. We apply the λ operator to the remaining closures and obtain the following datalog program:

$$TaxCut(x) \vee y_1 \approx y_2 \leftarrow hasChild(x, y_1), hasChild(x, y_2) \tag{29}$$
$$TaxCut(x) \vee x_f \approx y_2 \leftarrow Q_1(x), hasChild(x, y_2), S_f(x, x_f) \tag{30}$$
$$\leftarrow Man(x), Woman(x) \tag{31}$$
$$Woman(x) \leftarrow hasDaughter(x, y) \tag{32}$$
$$Q_2(x_f) \leftarrow Q_1(x), S_f(x, x_f) \tag{33}$$
$$Woman(x) \leftarrow Q_2(x) \tag{34}$$

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

$$Q_1(Peter) \tag{35}$$
$$hasChild(Peter, Paul) \tag{36}$$
$$Man(Paul) \tag{37}$$
$$S_f(Peter, Peter_f) \tag{38}$$
$$S_f(Paul, Paul_f) \tag{39}$$

Now to answer the query $?-TaxCut(\mathtt{x})$, we make all literals involving the predicate $TaxCut$ smallest in the ordering. Next, we hyperresolve (30) with (35), (36) and (38) to obtain (40). Next we hyperresolve (33) and (35) to obtain (41), which is then hyperresolved with (34) to obtain (42). Now (40) is superposed into (42) to obtain (43). Finally, we hyperresolve (31) with (42) and (43) to obtain the desired conclusion (44).

$$TaxCut(Peter) \vee Peter_f \approx Paul \tag{40}$$
$$Q_2(Peter_f) \tag{41}$$
$$Woman(Peter_f) \tag{42}$$
$$TaxCut(Peter) \vee Woman(Paul) \tag{43}$$
$$TaxCut(Peter) \tag{44}$$

7 Conclusion

Motivated by the prospects of optimizing ABox reasoning by reusing optimization techniques from disjunctive deductive databases, in this paper we present a technique for reducing $\mathcal{ALCHIQ}^-$ knowledge bases to disjunctive datalog programs. In order to do that, we devise a decision procedure for satisfiability checking based on basic superposition. Our decision procedure runs in EXPTIME in the size of the knowledge base for unary coding of numbers in the input.

Next we show how to use this procedure to reduce any $\mathcal{ALCHIQ}^-$ knowledge base to a disjunctive datalog program. This is possible because we can safely remove from the saturated set of clauses produced by our procedure all clauses containing functional terms of depth greater than one, and simulate the ground functional terms of depth one with fresh constants.

We demonstrate that any $\mathcal{SHIQ}$ knowledge base can be polynomially encoded as an $\mathcal{ALCHIQ}$ knowledge base without affecting satisfiability. Thus, our approach provides means for handling a great number of interesting description logics. In particular, it is useful in the Semantic Web context, since it can handle most of OWL-DL, with the exception of of nominals and the restriction to very simple roles. On the other hand, we support qualified number restrictions, which are not present in OWL-DL.

We believe that our approach will enable efficient ABox reasoning primarily because reduction to disjunctive datalog allows us to use various optimizations, such as join order optimizations or the magic sets transformation. The latter has been show to dramatically improve the evaluation of disjunctive datalog programs, as it reduces the number of models of the disjunctive program.

For our future work, we see five theoretical and one practical challenge. The theoretical challenges are: dropping the constraint on very simple roles, extending the logic with nominals, providing a decision procedure in EXPTIME regardless of the coding of numbers, investigating whether basic superposition can be used to decide answering conjunctive queries over $\mathcal{ALCHIQ}$ knowledge bases, and providing support for reasoning with data types.

From the practical point of view, we intend to implement a new description logic inference system, compare its performance to state-of-the-art systems, and thus validate our approach in practice.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. L. Bachmair and H. Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
4. L. Bachmair and H. Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel and P. H. Schmidt, editors, *Automated Deduction: A Basis for Applications*, volume I, Foundations: Calculi and Methods, chapter 11. Kluwer Academic Publishers, Dordrecht, 1998.
5. L. Bachmair and H. Ganzinger. Strict Basic Superposition. In *Automated Deduction—CADE-15*, volume 1421 of *Lecture Notes in Computer Science*, pages 160–174, Lindau, Germany, July 1998. Springer.
6. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
7. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.
8. C. Beeri and R. Ramakrishnan. On the power of magic. In *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, pages 269–293, San Diego, CA, March 1987.

9. S. Brass and U. W. Lipeck. Generalized Bottom-Up Query Evaluation. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology - Proceedings EDBT'92*, pages 88–103, Berlin, 1992. Springer-Verlag.
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. Elsevier Science, 2001.
11. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
12. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-Monotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 364–375, Dagstuhl, Germany, July 1997. Springer.
13. M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Springer Verlag, 1996.
14. H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proc. of the 14th IEEE Symposium on Logic in Computer Science*, pages 295–305. IEEE Computer Society Press, 1999.
15. G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *J. of the ACM*, 32(2):280–295, 1985.
16. S. Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Trans. on Knowledge and Data Engineering*, 15(2):717–736, March/April 2003.
17. B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.
18. V. Haarslev and R. Möller. Optimization Strategies for Instance Retrieval. In *Proc. Int'l Workshop on Description Logics (DL-2002)*, Toulouse, France, April 2002.
19. V. Haarslev and R. Möller. Incremental Query Answering for Implementing Document Retrieval Services. In D. Calvanese, G. de Giacomo, and F. Franconi, editors, *Proc. Int'l Workshop on Description Logics (DL-2003)*, pages 85–94, Rome, Italy, September 2003.
20. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
21. William McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.
22. B. Motik, A. Maedche, and R. Volz. Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases. In *10th International Workshop on Knowledge Representation meets Databases (KRDB-2003)*, Hamburg, Germany, September 15-16 2003.
23. B. Nebel. Terminological Cycles: Semantics and Computational Properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.
24. R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Logic and Computation*, 19(4):312–351, April 1995.
25. H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, 8(3):265–292, May 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. Elsevier Science, 2001.
27. P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. OWL Web Ontology Language; Semantics and Abstract Syntax, W3C Candidate Recommendation. http://www.w3.org/TR/owl-semantics/, November 2002.

28. D. A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Transformation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.
29. R. A. Schmidt and U. Hustadt. A Principle for Incorporating Axioms into the First-Order Translation of Modal Formulae. In F. Baader, editor, *Automated Deduction—CADE-19*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 412–426. Springer, 2003.
30. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.