Declarative PTIME Queries for Relational Databases using Quantifier Elimination

PATRICK DOHERTY, *Department of Computer and Information Science, Linköping University, Linköping, Sweden.*
E-mail: patdo@ida.liu.se

WITOLD ŁUKASZEWICZ and ANDRZEJ SZAŁAS, *Institute of Informatics, Warsaw University, ul. Banacha 2, 02-097 Warsaw, Poland.*
E-mail: $\{$*witlu,szalas*$\}$*@mimuw.edu.pl*

Abstract

In this paper, we consider the problem of expressing and computing queries on relational deductive databases in a purely declarative query language, called SHQL (Semi-Horn Query Language). Assuming the relational databases in question are ordered, we show that all SHQL queries are computable in PTIME (polynomial time) and the whole class of PTIME queries is expressible in SHQL. Although similar results have been proven for fixpoint languages and extensions to datalog, the claim is that SHQL has the advantage of being purely declarative, where the negation operator is interpreted as classical negation, mixed quantifiers may be used and a query is simply a restricted first-order theory not limited by the rule-based syntactic restrictions associated with logic programs in general. We describe the PTIME algorithm used to compute queries in SHQL which is based in part on quantifier elimination techniques and also consider extending the method to incomplete relational databases using intuitions related to circumscription techniques.

Keywords: Deductive databases, relational databases, query languages, second-order quantifier elimination.

1 Introduction

In this paper, we consider the problem of expressing and computing queries on relational deductive databases in a purely declarative query language we introduce, called SHQL (Semi-Horn Query Language). The language is declarative in the sense that queries are expressed in classical logic, having a well-defined semantics which does not refer to any aspect of the underlying execution mechanism.

Assuming the relational databases in question are ordered, we show that all SHQL queries are computable in PTIME[1] and the whole class of PTIME queries is expressible in SHQL. Similar results have been proven for fixpoint languages and extensions to datalog, but the claim will be that these languages are not purely declarative and that SHQL provides a more natural means of expressing queries.

Much recent activity in the area of deductive databases has focused on the language *datalog* and its extensions which integrate recursion with negation [1]. When adding negation to datalog, this requires defining a semantics for negative facts. There are many choices as to such a semantics and these choices influence not only the natural interpretation of the

[1] In this paper, PTIME stands for polynomial time.

negation symbol in a query, but the expressiveness of the language. For example, stratified semantics requires syntactic restrictions on the use of negation in a datalog$^{\neg}$ program, while well-founded semantics, although not requiring syntactic restrictions, does use a 3-valued semantics to interpret the meaning of a program. In addition, while well-founded semantics is equivalent to the fixpoint queries, stratified semantics is strictly weaker.

An important aspect of query language design is to achieve a good balance between the expressiveness of the language and the complexity of evaluating queries in the language. In addition to expressiveness and efficiency, the language should be natural to use. Although it can be argued that extended datalog languages achieve the goals of expressiveness and efficiency in theory, one can debate the naturalness of using datalog as a query language. For instance, the variations in interpretation already discussed can be quite confusing for a normal user of the query language. The procedural leakage into the language resulting from the use of alternative non-classical interpretations of the negation operator and the various syntactic restrictions such choices place on datalog programs, tends to violate the basic tenets of a good declarative query language.

On the other hand, SHQL is a purely declarative query language. Use of negation in a query is interpreted as classical negation, a class of mixed quantifiers is allowed in queries, and intentional and extensional predicates may occur anywhere in the query. SHQL is not rule-based and a query is expressed as a theory consisting of semi-Horn formulas (defined in Section 5.1).

SHQL is used as follows. Given the task of computing a definition of an intensional predicate Q (or asking whether a tuple is an instance of Q) relative to a relational database B consisting of the relations $R_1, \ldots, R_n$, we first provide an implicit definition of Q in terms of a SHQL theory, $\Theta(Q)$, which is essentially a conjunction of semi-Horn formulas using any of $R_1, \ldots, R_n$, and Q. The theory $\Theta(Q)$ is only constrained by the fact that it must be semi-Horn. All quantifiers and logical connectives are interpreted classically. The goal is to compute an explicit definition of Q in PTIME which is interpreted as the result of the query $\Theta(Q)$.

The computation process can be described in two stages. In the first stage, we provide a PTIME (in the size of the input query) compilation process which uses a quantifier elimination algorithm called the DLS algorithm [6]. An extension for fixpoint formulas is called the G-DLS algorithm [5, 4, 7]. The DLS algorithm takes as input a second-order formula and returns a logically equivalent first-order formula, or terminates with failure, where failure does not mean there is not a reduction, but simply that the algorithm cannot find one. The G-DLS algorithm is a generalization of the DLS algorithm and returns logically equivalent fixpoint formulas for a wider class of inputs. Both algorithms can be combined into one algorithm which we denote by DLS* (see [4, 7]). Given the SHQL query, $\Theta(Q)$, we prefix it with an existential quantifier and input the formula $\exists Q.\Theta(Q)$ to DLS*. If the query is first-order definable, then the output will be a logically equivalent first-order formula expressing an explicit definition of Q. The output is computed in PTIME and LOGSPACE (in the size of the database). If the query is not first-order definable, then the output will be a logically equivalent fixpoint formula expressing an explicit definition of Q. In this case, output is computed in PTIME. Note that this technique can be used for theories outside the semi-Horn class, but neither the complexity results nor a successful reduction are guaranteed.

In the second stage, we use the explicit definition of Q (output in the first stage) to compute a suitable relation in the relational database that satisfies Q. Before computing the output relation, we first check to see that such a relation exists relative to the database. Suppose $\Theta(Q)$

is the original query, B the relational database and $\Theta'(Q)$ the output of DLS* given the input $\exists Q.\Theta(Q)$. We say that the query $\Theta(Q)$ is a *coherent query relative to* B if $B \models \Theta'(Q)$. Assuming this is the case, we know that the output relation exists and can now compute the answer. Both checking that the query is coherent ($B \models \Theta'(Q)$) and computing the output relation can be done efficiently because calculating fixpoint queries and fixpoint satisfiability checking over finite domains are both in PTIME (see Immerman [10], Sazonov [13], Vardi [15]).

Note that although the combined problem of finding out whether an implicit query $\Theta(Q)$ to a database exists, checking that the query is coherent, and explicitly computing the answer is in general NP-complete (in the size of the database), as was shown by Fagin (see Immerman [9]), our method which applies quantifier elimination techniques to semi-Horn theories makes the problem solvable in polynomial time for this special case. Most importantly, SHQL is an expressive language which covers all PTIME queries and is at the same time purely declarative. Querying with SHQL is as natural as querying with classical logic and the compilation step is completely transparent to the user.

The rest of the paper will be structured as follows. In Section 2 we introduce the concept using an introductory example. In Section 3, we provide preliminary definitions. In Section 4, we describe Ackermann's Lemma and the Fixpoint Theorem, which provide the formal basis for the DLS* algorithm. In fact, using the full algorithm is not necessary in order to achieve our goals. Some more direct syntactic manipulations together with these theorems provably achieve the same reduction results and are discussed in Section 5.2. In Section 5, we provide a detailed description of our two-stage method and discuss a technique which permits queries with more than one intensional predicate. In Section 6, we provide a number of examples which demonstrate the naturalness of SHQL and the proposed querying method. Finally, in Section 7, we consider an extension of the method to relational databases with incomplete information. We then conclude with a discussion. An appendix includes a detailed description of the DLS, G-DLS, and DLS* algorithms.

2 An introductory example

As we mentioned in Section 1, in order to retrieve information from a database, we describe the query in terms of the properties of a desired relation using semi-Horn formulas. The properties of the relation usually do not give us its explicit definition directly. Moreover, some new facts are deduced in order to calculate the output relation. Since we propose a new methodology, quite different from the currently accepted paradigm of selecting information by using SQL-like statements, we first provide an introductory example illustrating the methodology and consider potential problems which may arise.

EXAMPLE 2.1
Assume we are provided with a database of information about SOQT[2] Society members. There are two types of society members, the so called distinguished terminators and the ordinary terminators. A candidate for membership in the society is a person who is not in the society and is nominated by a distinguished terminator or at least two ordinary terminators. In addition, by a rule accepted by the society, candidates can be considered by a selection committee provided that at least one of the candidates is nominated by a distinguished terminator. The database contains information about members of the society and about candidates,

[2]SOQT stands for Second Order Quantifier Termination.

stored as the following extensional relations:

- $P(x)$, containing data about persons (for simplicity represented as x);
- $D(x)$, meaning that a person x is a distinguished terminator;
- $O(x)$, meaning that a person x is an ordinary terminator;
- $N(x, y)$, meaning that a person y is a candidate nominated by a person x.

We now want to select all candidates to the society and make sure that at least one candidate is nominated by a distinguished terminator. We thus want to calculate a (maximal) relation $C(x)$, meaning that x is a candidate, satisfying (the conjunction of) the following conditions:

1. $\forall x(C(x) \supset P(x))$ (any candidate is a person).
2. $\forall x(C(x) \supset (\neg D(x) \wedge \neg O(x)))$ (a candidate is not in the society already).
3. $\forall x(C(x) \supset (\exists y(D(y) \wedge N(y, x)) \vee \exists y \exists z(O(y) \wedge O(z) \wedge y \neq z \wedge N(y, x) \wedge N(z, x)))$ (a candidate must be nominated by a distinguished terminator or at least two ordinary terminators).
4. $\exists u \exists v(D(u) \wedge N(u, v) \wedge C(v))$ (there is a candidate nominated by a distinguished terminator).

The above specification directly reflects our rules as formulas of classical logic, i.e. is purely declarative. Next we try to calculate the output relation C. Unfortunately, we do not have an explicit definition of C in the database. Therefore, we cannot apply any SQL-like SELECT statement directly as it would require an explicit definition in the WHERE clause (translation of existential quantifiers would also be problematic here). We cannot use *datalog* as there are existential quantifiers in the scope of universal quantifiers. One cannot eliminate these quantifiers by Skolemization, since this move introduces function symbols that are not allowed in *datalog*. Similarly, fixpoint queries are not directly applicable here either. Another problem which appears is that formula (4) in the current example implies the following condition:

$$\exists u \exists v(D(u) \wedge N(u, v)).$$

The problem that arises when such constraints are deduced from queries also has to be addressed. Sometimes the constraints follow from more advanced deductions and should somehow be calculated. If we apply the method we propose, our query results in both the generation of an explicit definition of C and the additional constraints which we call *coherence conditions*. The generation of both the explicit definition and the coherence condition is computed in time polynomial in the size of the original query, provided it is formulated as a semi-Horn formula (for the definition see Section 5.1). Moreover, checking that the coherence condition is entailed by the database and calculating the output relation specified by a semi-Horn formula is done in time polynomial in the size of the database. Thus the method we propose is acceptable if a complexity argument is used as one of the criteria for evaluation of the method. In the example used, we deal with semi-Horn formulas where the explicit definition for C generated by the technique would be:

$$C(x) \equiv \quad [P(x) \wedge \neg D(x) \wedge \neg O(x) \wedge (\exists y(D(y) \wedge N(y, x)) \vee \exists y \exists z(O(y) \wedge O(z) \wedge y \neq z \wedge N(y, x) \wedge N(z, x)))]$$

and the coherence condition would be:

$$\exists u \exists v(D(u) \wedge N(u, v) \wedge C(v)),$$

where C is substituted by its explicit definition (with renaming of x by v).

It is worth emphasizing here that the coherence condition is 'local' to the query, i.e. a database may not be consistent together with a query but still itself have a consistent contents. For example, consider a database without any nominees (i.e. where the relation N is empty). Of course, the database may be consistent. However, when one asks the query about C, the coherence condition, which is a logical consequence of the query, is violated.

Such a phenomenon does not occur in other query languages known from the literature, including *datalog*, since queries in those languages cannot cause any inconsistencies not already present in a database.

3 Definitions

In what follows we deal with the full classical first-order logic where formulas are built from atoms (i.e. formulas of the form $R(\bar{t})$, where R is a relation symbol and $\bar{t}$ is a list of terms) according to the following usual rule:

$$\begin{array}{rl}\langle\text{Formula}\rangle ::= & \langle\text{Atom}\rangle \mid \neg\langle\text{Formula}\rangle \mid \langle\text{Formula}\rangle \circ \langle\text{Formula}\rangle \mid \\ & \forall\bar{x}\langle\text{Formula}\rangle \mid \exists\bar{x}\langle\text{Formula}\rangle,\end{array}$$

where $\circ \in \{\vee, \wedge, \supset, \equiv\}$ is a propositional connective (disjunction, conjunction, implication or equivalence, respectively). By a *theory* we always mean a finite set of axioms (formulas of the classical logic). Thus theories can be transformed into formulas (conjunctions of axioms).

DEFINITION 3.1
A *relational database* B, is a first-order structure

$$\langle U, r_1^{a_1}, \ldots, r_k^{a_k}, c_1, \ldots, c_l\rangle,$$

where

- U is a finite set,
- for $1 \leq i \leq k$, $r_i^{a_i}$ is an a_i-ary relation on U, i.e. $r_i^{a_i} \subseteq U^{a_i}$, and
- $c_1, \ldots, c_l \in U$ are constants.

By a *signature* of B we mean a signature containing relation symbols $R_1^{a_1}, \ldots, R_k^{a_k}$ and constant symbols $C_1, \ldots, C_l$ together with equality $=$.

According to accepted terminology in the literature (introduced in [12]), a deductive database consists of two parts: an extensional and intensional database. The extensional database is usually equivalent to a traditional relational database (without views) and the intensional database contains a set of definitions of relations that are not explicitly stored in the database. Accordingly, we have the following definition.

DEFINITION 3.2
By a *deductive database* we understand a relational database augmented with an additional set of formulas defining fresh relations in terms of a chosen logic. The relational database is called *extensional* and the set of formulas is called an *intensional database*. We say that a relation (relation symbol) is *intensional* in a database if it appears in the intensional database only, otherwise it is called *extensional*.

DEFINITION 3.3
We say that a formula Φ is *positive* w.r.t. a predicate P iff P appears under no negation sign in Φ (in negation normal form[3]). Dually, we say that Φ is *negative* w.r.t. P iff all occurrences of P have the form $\neg P$ and $\neg P$ appears under no negation sign in Φ.

DEFINITION 3.4
Let $\mathcal{L}_{\mathcal{I}}$ be the classical first-order logic and σ be a signature. By the *fixpoint calculus* over σ, denoted by $\mathcal{L}^{\sigma}_{\mathcal{F}}$, we understand this to be the logic obtained from $\mathcal{L}_{\mathcal{I}}$ by extending it with the least fixpoint operator $\mu P.\Phi(P)$, where Φ is positive w.r.t. P. We define $\nu P.\Phi(P)$ as $\neg\mu\neg P.\neg\Phi(P)$.[4]

Note that $\mu P(\bar{x}).\Phi(P)$ is the least (w.r.t. implication) formula $\Psi(\bar{x})$ such that

$$\Psi(\bar{x}) \equiv \Phi(P \leftarrow \Psi(\bar{x})).$$

Every formula $\Phi(P)$ which is positive w.r.t. P is monotone and therefore, by the Knaster and Tarski fixpoint theorem, the fixpoints we consider are well defined.

DEFINITION 3.5
Let $B = \langle U, r_1^{a_1}, \ldots, r_k^{a_k}, c_1, \ldots, c_l \rangle$ be a relational database and let σ be a signature of B.

- By a *first-order query language for* B, denoted by $\mathcal{L}^{\mathcal{B}}_{\mathcal{I}}$ we mean the classical first-order logic over signature σ.
- By a *fixpoint query language for* B, denoted by $\mathcal{L}^{\mathcal{B}}_{\mathcal{F}}$ we mean the fixpoint calculus over signature σ.
- By an *implicit query to* B we mean a classical first-order formula $\Theta(Q)$ over signature σ augmented with an additional relation symbol Q (representing the relation to be calculated).

Observe that in the case of an implicit query, say $\Theta(Q)$, it is natural to demand that Q represents the minimal or maximal relation s satisfying Θ, provided that such s exists.[5] Let us note that maximizing a relation corresponds to minimizing its complement.[6] Accordingly, and without loss of generality, we shall focus on minimizing relations.

DEFINITION 3.6
Let $B = \langle U, r_1^{a_1}, \ldots, r_k^{a_k}, c_1, \ldots, c_l \rangle$ be a relational database.

- The semantics of the query language $\mathcal{L}^{\mathcal{B}}_{\mathcal{I}}$ is defined as in the case of first-order logic, assuming that for $1 \le i \le k$, relation symbols $R_i^{a_i}$ are interpreted as relations $r_i^{a_i}$ and for $1 \le j \le l$, constant symbols C_j are interpreted as constants c_j.
- The semantics of the query language $\mathcal{L}^{\mathcal{B}}_{\mathcal{F}}$ is defined by extending the definition of the semantics of $\mathcal{L}^{\mathcal{B}}_{\mathcal{I}}$, assuming that $\mu Q.\Phi(Q)$ represents the least (w.r.t. $\subseteq$) relation s such that $B \models Q = \Phi(Q)$ with Q interpreted as s.
- The semantics of an implicit query $\Theta(Q)$ is defined as the least (w.r.t. $\subseteq$) relation s such that $B \models \Theta(Q)$ with Q interpreted as s, provided that such s exists. A formula expressing the existence of s is called the *coherence condition* for $\Theta(Q)$.

[3]That is, in a form where negation can appear only before atoms.
[4]Observe that we use a notation, where the fixpoint is applied to a negated variable. A more readable form of this formula can be obtained by replacing P by $\neg P$ and writing $\neg\mu P.\neg\Phi(\neg P)$.
[5]The existence of s means that $\Theta(Q)$ is consistent with the database.
[6]This does not have to hold when a database is allowed to contain incomplete information—see Section 7.

It is often convenient to reference some ‘columns’ of a relation in a database. We often do this by extending the signature with relation symbols corresponding to columns. For example, given a relation $person \subseteq String \times Integer$, we might want to refer to the first column using the predicate symbol $Name$ where $Name(x, john)$ means that the name of person x is $john$. If a column, being represented by, say C, contains Boolean values then we write $C(x)$ to mean $C(x, \top)$ and $\neg C(x)$ to mean $C(x, \bot)$. This notation for referencing columns will prove to be useful in Section 7, where we consider incomplete databases.

4 Ackermann’s Lemma and a fixpoint theorem

In the introduction, we stated that the DLS* algorithm was a combination of two separate algorithms, DLS and G-DLS. DLS* works as follows. The input to DLS* is first passed to the DLS algorithm. If the input can be put into what we call an *Ackermann reducible formula* (for the definition of those formulas see Section 5.1) then DLS* outputs a logically equivalent first-order formula. If the DLS algorithm terminates with failure, then the input is passed to the G-DLS algorithm. If the input can be put into what we call a *semi-Horn formula* then the DLS* algorithm outputs a fixpoint formula logically equivalent to the input. Of course, certain optimizations can be made which combine the two algorithms in a more efficient manner. The basis for both the DLS and G-DLS algorithms are two theorems which we describe below. Both provide a means of eliminating quantifiers which bind predicate variables. These two theorems provide the formal basis for the compilation step described previously and called stage one, where an SHQL query is first prefixed with an existential quantifier which binds the intensional predicate whose explicit definition we would like to generate and compute. The second-order query is then passed to the DLS* algorithm. For a detailed description of the DLS* algorithm, see the Appendix.

The following lemma was proved by Ackermann in [2] (for an alternative proof see also [14]).

LEMMA 4.1
Let P be a predicate variable and $\Phi(\bar{x}, \bar{z})$, $\Psi(P)$ be formulas without second-order quantification. Let Φ contain no occurrences of P at all. Then the following equivalences hold:

Let $\Psi(\neg P)$ be negative w.r.t. P, then

$$\exists P \forall \bar{x}[P(\bar{x}) \vee \Phi(\bar{x}, \bar{z})] \wedge \Psi(\neg P) \ \equiv\ \Psi(\neg P \leftarrow \Phi(\bar{x}, \bar{z})). \tag{4.1}$$

Let $\Psi(P)$ be positive w.r.t. P, then

$$\exists P \forall \bar{x}[\neg P(\bar{x}) \vee \Phi(\bar{x}, \bar{z})] \wedge \Psi(P) \ \equiv\ \Psi(P \leftarrow \Phi(\bar{x}, \bar{z})), \tag{4.2}$$

where in the right-hand formulae the arguments $\bar{x}$ of Φ are to be substituted by the respective actual arguments of P (renaming the bound variables whenever necessary).

The following theorem, extending Lemma 4.1, is proved in [11].

THEOREM 4.2
Assume that all occurrences of the predicate symbol P in the formula Ψ have only variables as arguments. Then the following equivalences hold:

Let $\Phi(\neg P)$ and $\Psi(\neg P)$ be negative w.r.t. P, then

$$\exists P \forall \bar{y}[P(\bar{y}) \vee \Phi(\neg P)] \wedge [\Psi(\neg P)] \ \equiv\ \Psi[\neg P \leftarrow \nu \neg P(\bar{y}).\Phi(\neg P)]. \tag{4.3}$$

Let $\Phi(P)$ and $\Psi(P)$ be positive w.r.t. P, then

$$\exists P \forall \bar{y}[\neg P(\bar{y}) \vee \Phi(P)] \wedge [\Psi(P)] \equiv \Psi[P \leftarrow \nu P(\bar{y}).\Phi(P)], \tag{4.4}$$

where the above substitutions exchange the variables bound by fixpoint operators by the corresponding actual variables of the substituted predicate.

Formula (4.2) of Lemma 4.1 and formula (4.4) of Theorem 4.2 are applied in the case of maximizing relations, while formulas (4.1) and (4.3) are applied when minimizing relations.

Lemma 4.1 is subsumed by Theorem 4.2. Moreover, any fixpoint formula of the form $\nu P.\Psi$, where Ψ does not contain P, is equivalent to Ψ. Thus one can, in all cases, use Theorem 4.2 and simplify the resulting formulas by applying this equivalence. This optimization diverges from the conceptual description we have been using when describing the DLS* algorithm, but it will simplify the detailed description of the query method described in the next section.

5 The method

We first observe that the problem whether a result of an implicit query $\Theta(Q)$ to a database B exists reduces to the question whether the second-order formula $\exists Q \Theta(Q)$ is satisfied in B. By Fagin's theorem the problem is NP-complete in the size of B (see [9]). In what follows we concentrate on selecting a class of implicit queries for which the problem is in $PTIME$.

Conceptually, the SHQL query method consists of four steps:

1. State a query $\Theta(Q)$ on a relational database B in SHQL, where $\Theta(Q)$ is a semi-Horn formula (for the definition of semi-Horn formulas see Section 5.1). Prefix the query with an existential quantifier binding the intensional predicate whose implicit definition in terms of $\Theta(Q)$ we would like to make explicit. The input to the compilation stage is $\exists Q.\Theta(Q)$.
2. Pass the input $\exists Q.\Theta(Q)$ to the DLS* algorithm. Assuming the input is semi-Horn, the algorithm will return either a logically equivalent first-order formula or a fixpoint formula. Call the output $\Theta'(Q)$.
3. Before explicitly computing the answer to the original query $\Theta(Q)$, check to make sure the query is *coherent* relative to B. We do this by essentially checking that $\Theta'(Q)$ is satisfied by B. If the query is not coherent, the algorithm produces no result (the user should be informed about the situation and perhaps be supplied with those database elements/rows that caused the inconsistency).
4. If the query is coherent, then compute the definition of Q (or check whether a tuple belongs to Q).

Provided the input is semi-Horn, all steps in the method can be computed in PTIME. In the following subsections, we will formally define the query language, describe and justify each of steps 2–4 with appropriate theorems, and conclude with a representation theorem characterizing the expressiveness and descriptive complexity of the query language.

5.1 The semi-Horn query language (SHQL)

In previous sections, we discussed SHQL informally, stating that queries to the database had to be in what we called *semi-Horn* form. In practice, we can do more. A query $\Theta(Q)$

can be any formula in a classical first-order language. If it is, or can be transformed into, semi-Horn form, then the DLS* algorithm is guaranteed to generate an explicit definition of Q which is logically equivalent to $\exists Q.\Theta(Q)$. If it is not, the DLS* algorithm may still terminate successfully and steps 2–4 above would still apply. In this section, we will make these intuitions precise.

We shall consider two types of formulas of the form

$$\Phi_1(Q) \wedge \Phi_2(Q), \tag{5.1}$$

where $\Phi_2(Q)$ is any first-order formula negative w.r.t. Q. We call these two types Ackermann-reducible formulas and semi-Horn formulas. They are defined as follows:

- *Ackermann-reducible formulas (w.r.t. Q)* are of the form (5.1) for which $\Phi_1(Q)$ is a conjunction of formulas of the form $\forall\bar{x}(Q(\bar{t})\vee\Psi)$, where Ψ is an arbitrary Q-free first-order formula.
- *Semi-Horn formulas (w.r.t. Q)* are of the form (5.1) for which $\Phi_1(Q)$ is a conjunction of formulas of the form $\forall\bar{x}(Q(\bar{t})\vee\Psi(\neg Q))$, and Ψ is an arbitrary first-order formula negative w.r.t. Q.

The negative dual forms are obtained by substituting Q by $\neg Q$ in the definitions, making Ψ an arbitrary first-order formula positive w.r.t. Q, and making $\Phi_2(Q)$ negative w.r.t. Q. In this case we are able to find the greatest solution for $\neg Q$, that is, a minimal solution for Q.

Ackermann-reducible formulas are also semi-Horn formulas, since in semi-Horn formulas no occurrence of $\neg Q$ in Ψ is required. However, it is important to isolate this class of formulas because these define first-order expressible queries. If the initial query $\Theta(Q)$ can be transformed into this form, then in the compilation step where $\exists Q.\Theta(Q)$ is given as input to the DLS* algorithm, the call to DLS will return a logically equivalent first-order formula. In fact, the DLS* algorithm, when successful, basically transforms $\Theta(Q)$ into one of the above forms (or their negative duals). Consequently, it is important to note that the method can be made more general by generating solutions for arbitrary formulas $\Gamma(Q)$ which although not semi-Horn, are reducible by the DLS* algorithm.

Note that any conjunction of semi-Horn formulas w.r.t. Q can be transformed into the following form:

$$\forall\bar{x}[\Phi(\bar{x},\overline{z_i},Q)\supset Q(\bar{x})]\wedge\Psi(\neg Q), \tag{5.2}$$

where $\Psi(\neg Q)$ is an arbitrary first-order formula negative w.r.t. Q and $\Phi(\bar{x},\overline{z_i},Q)$ is positive w.r.t. Q, or its dual,

$$\forall\bar{x}[\Phi(\bar{x},\overline{z_i},\neg Q))\supset\neg Q(\bar{x})]\wedge\Psi(Q), \tag{5.3}$$

where $\Psi(Q)$ is an arbitrary first-order formula positive w.r.t. Q and $\Phi(\bar{x},\overline{z_i},\neg Q)$ is negative w.r.t. Q. To see this, it suffices to use the following equivalence that allows us to combine two semi-Horn formulas into a single semi-Horn formula:

$$\begin{aligned}&[\forall\bar{x}(\Phi(\bar{x},\overline{z_i},Q)\supset Q(\bar{x}))\wedge\Psi(\neg Q)]\wedge[\forall\bar{x}(\Phi'(\bar{x},\overline{z'_i},Q)\supset Q(\bar{x}))\wedge\Psi'(\neg Q)]\\ &\quad\equiv[\forall\bar{x}((\Phi(\bar{x},\overline{z_i},Q)\vee\Phi'(\bar{x},\overline{z'_i},Q))\supset Q(\bar{x}))\wedge(\Psi(\neg Q)\wedge\Psi'(\neg Q))].\end{aligned} \tag{5.4}$$

A similar equivalence applies for the dual forms. Moreover, if we have many intensional predicates, we can easily encode these by a single predicate which has a vector of additional Boolean variables distinguishing between various relations. For instance, if we have two predicates, say $P(\bar{x})$ and $Q(\bar{y})$, then we can use a single predicate $R(z,\bar{x},\bar{y})$ such that

$R(\top, \bar{x}, \bar{y})$ means $P(\bar{x})$ and $R(\bot, \bar{x}, \bar{y})$ means $Q(\bar{y})$. The resulting formulas are still semi-Horn formulas. Thus, we can safely assume that we always deal with a single intensional predicate in our queries.

It is worth emphasizing here that semi-Horn formulas are strictly more expressive than Horn clauses. For instance, semi-Horn formulas express the complement of a relation which is not expressible by Horn clauses [3]. The semi-Horn (w.r.t. Q) formula

$$\forall \bar{x}(Q(\bar{x}) \vee R(\bar{x})) \wedge \forall \bar{x}(\neg Q(\bar{x}) \vee \neg R(\bar{x}))$$

expresses the complement of a relation, R. In addition, existential quantifiers in the scope of universal quantifiers are not, in general, reducible to Horn clauses, but are allowed in semi-Horn formulas.

Let us now introduce the definition of declarative queries and declarative query language. As we shall see in Theorem 5.4, all the queries of the language are computable in polynomial time. Moreover, the whole class of PTIME queries is covered by the language.

DEFINITION 5.1
By a *declarative query* we mean any implicit query expressed as a semi-Horn formula. By a *declarative query language SHQL* we mean a first-order query language augmented with declarative queries, assuming that the underlying signature contains a relation that, on the semantic side, linearly orders domains of databases.

5.2 The compilation, coherence, and computing steps

It is easily observed that Ackermann-reducible formulas are reducible to classical first-order formulas by applying the DLS algorithm which is based on Lemma 4.1 and that semi-Horn formulas are reducible to fixpoint formulas by applying the G-DLS algorithm which is based on Theorem 4.2 [6, 5].

We also observe two additional facts concerning Lemma 4.1 and Theorem 4.2. Namely, assume we are given a formula $\exists Q \Phi(Q)$. By the proofs of Lemma 4.1 and Theorem 4.2, where reduction is successful (which is always the case for semi-Horn queries) one gets:

- a first-order (or fixpoint) definition of Q (this definition is used in suitable substitutions in the resulting formulas), and
- a first-order (or fixpoint) formula equivalent to the input formula.

The first observation justifies the generation of a fixpoint or first-order formula that explicitly defines the query. The second observation justifies the generation of the coherence condition for a query expressed as a fixpoint or first-order formula. Note that the explicit definition and coherence condition are either both first-order or both fixpoint formulas. The coherence condition allows us to check whether the output relation exists. If we know that the output relation exists, we can calculate the answer using the formula obtained via the first observation. Both the coherence check and calculation of the output relation can be done in polynomial-time by using an algorithm for calculating fixpoint queries and for checking fixpoint satisfiability over finite domains which is described in [10]. The process of calculating the output relation and checking coherence can be optimized by noting that more or less the same fixpoint formula appears in the explicit definition and the coherence condition.

The following theorem easily follows from a result in [5].

THEOREM 5.2
For any formula $\Theta(Q)$ of the form (5.2):

- the explicit definition of Q is given by $Q(\bar{x}) \equiv \mu Q(\bar{x}).\Phi(\bar{x}, \overline{z_i}, Q)$, and
- the coherence condition for $\Theta(Q)$ is $\Psi(Q \leftarrow \mu Q(\bar{x}).\Phi(\bar{x}, \overline{z_i}, Q))$.

For any formula $\Theta(Q)$ of the form (5.3):

- the explicit definition of Q is given by $Q(\bar{x}) \equiv \nu Q(\bar{x}).\neg\Phi(\bar{x}, \overline{z_i}, \neg Q)$, and
- the coherence condition for $\Theta(Q)$ is $\Psi(Q \leftarrow \nu Q(\bar{x}).\neg\Phi(\bar{x}, \overline{z_i}, \neg Q))$.

As a consequence we have the following theorem.

THEOREM 5.3
For any formula $\Theta(Q)$ of the form (5.2), where Φ does not contain Q:

- the explicit definition of Q is given by $Q(\bar{x}) \equiv \Phi(\bar{x}, \overline{z_i})$, and
- the coherence condition for $\Theta(Q)$ is $\Psi(Q \leftarrow \Phi(\bar{x}, \overline{z_i}))$.

For any formula $\Theta(Q)$ of the form (5.3), where Φ does not contain Q:

- the explicit definition of Q is given by $Q(\bar{x}) \equiv \neg\Phi(\bar{x}, \overline{z_i})$, and
- the coherence condition for $\Theta(Q)$ is $\Psi(Q \leftarrow \neg\Phi(\bar{x}, \overline{z_i}))$.

5.3 A representation theorem

For the declarative query language defined in Definition 5.1 we have the following theorem.

THEOREM 5.4
Let B be a relational database.

- Any implicit query $\Theta(Q)$ to B, where $\Theta(Q)$ is a semi Horn formula, is computable in polynomial time in the size of the database. Consequently, any query expressed in the declarative query language defined in Definition 5.1 is in PTIME. If $\Theta(Q)$ is an Ackermann reducible formula, then Q is computable in logarithmic space.
- Any PTIME query can be expressed in the declarative query language defined in Definition 5.1 provided that the domain of B is linearly ordered.

PROOF. The first part of the theorem can be proved by noticing that semi-Horn formulas reduce to fixpoint formulas and using the well-known fact that fixpoint queries are computable in polynomial time (see e.g. [3, 10]). If $\Theta(Q)$ is an Ackermann reducible formula, then both Q and the coherence condition are expressed in classical first-order logic and therefore are computable in logarithmic space.

To prove the second part of the theorem we use the following known facts:

- if the domain of the database is linearly ordered then fixpoint queries express all PTIME queries (see [3, 10, 13, 15]);
- under the above assumption all fixpoint queries can be expressed by taking a single fixpoint of a first-order formula with positive occurrences of the calculated predicate (followed by classical first-order operations) (see [3, 8, 10]).

According to the above, it is sufficient to prove that a single fixpoint can be defined by an implicit query. Assume that the fixpoint to be defined is $\mu P(\bar{x}).\Psi(P)$, where Ψ is positive w.r.t. P. By Theorem 5.2, the implicit query that defines this fixpoint is then simply the formula $\forall\bar{x}(\Psi(P) \supset P(\bar{x}))$. ■

It is also worth emphasizing here that in the case when a query is expressed as a semi-Horn theory, then both the size of the resulting explicit definition and the size of the coherence condition are polynomial in the size of the input query. In fact, from Theorems 5.2 and 5.3, it easily follows that the size of the explicit definition is linear in the size of the query (usually less than the size of the query) and the size of the coherence condition is less than the square of the size of the query.

6 Examples

In this section, we provide a number of examples which demonstrate both the expressiveness of semi-Horn queries and how the formal results may be applied practically.

EXAMPLE 6.1
This example demonstrates how the intensional predicate Q and the extensional predicates R, S, and E may be used anywhere in the query. In particular, in comparison with rule-based query languages such as logic programming or datalog, both intensional and extensional predicates may occur in both the head and body of any implication.

Assume we have a database B, containing information about whether persons are *rich*, *smart*, or *experienced*, denoted by the unary extensional predicates, R, S, and E, respectively. Suppose we are interested in selecting all rich persons and perhaps some others and we only want to consider those who are smart or experienced. Let Q denote the unary intensional predicate that describes the required relation. The first condition is then expressed by the formula

$$\forall x(R(x) \supset Q(x)),$$

while the second condition is expressed by the formula

$$\forall x(Q(x) \supset (S(x) \vee E(x)).$$

The implicit query $\Theta(Q)$ is then defined as the conjunction of the above formulas, where we are interested in obtaining the greatest relation Q satisfying

$$\forall x(Q(x) \supset (S(x) \vee E(x))) \wedge \forall x(R(x) \supset Q(x)).$$

After removing the implication sign, we have the equivalent,

$$\forall x(\neg Q(x) \vee (S(x) \vee E(x))) \wedge \forall x(\neg R(x) \vee Q(x)).$$

In order to maximize Q we will minimize its negation by using the dual form (5.3). $\Theta(Q)$ can be rewritten as $\Theta(\neg Q)$,

$$\forall x(\neg(S(x) \vee E(x)) \supset \neg Q(x)) \wedge \forall x(\neg R(x) \vee Q(x)),$$

which is easily observed to be of the form (5.3), where $\Phi(\bar{x})$ is $\neg(S(x) \vee E(x))$ and $\Psi(Q)$ is $\forall x(\neg R(x) \vee Q(x))$.

According to Theorem 5.3 the following formula of the form $\Psi(Q \leftarrow \neg\Phi(\bar{x}))$ is the suitable coherence condition:

$$\forall x(\neg R(x) \vee (S(x) \vee E(x))).$$

Observe that our query forces this condition (by transitivity of implication), Thus, for instance, if a database contains an element e such that $R(e)$ and $\neg S(e)$ and $\neg E(e)$, then the query is inconsistent with the database.

Now from Theorem 5.3 we obtain the explicit definition of $\neg Q$, which is $\neg Q(x) \equiv \neg(S(x) \vee E(x))$. Consequently, the explicit definition for Q is:

$$Q(x) \equiv S(x) \vee E(x).$$

EXAMPLE 6.2
Let E, R, S and Q be as in Example 6.1. Suppose we are now again interested in selecting rich persons and perhaps some others, but rule out those which are smart. In addition we want to make sure that at least one experienced person is selected. The implicit query $\Theta'(Q)$ can be expressed by formula

$$\forall x(R(x) \supset Q(x)) \wedge \forall z(S(z) \supset \neg Q(z)) \wedge \exists y(E(y) \wedge Q(y)).$$

The query is already in the form (5.3). The application of Theorem 5.3 shows that the coherence condition is

$$\forall x(R(x) \supset \neg S(x)) \wedge \exists y(E(y) \wedge \neg S(y))$$

and the explicit definition of $Q(x)$ is

$$Q(z) \equiv \neg S(z).$$

EXAMPLE 6.3
Consider the database containing a binary relation R. The following implicit query $\Theta(S)$ defines S as the transitive closure of R:

$$\forall x \forall y(R(x,y) \supset S(x,y)) \wedge \forall x \forall y \forall z((S(x,y) \wedge S(y,z)) \supset S(x,z)).$$

The above query is equivalent to the following formula:

$$\exists S[\forall x \forall z(S(x,z) \vee (\neg R(x,z) \wedge \forall y(\neg S(x,y) \vee \neg S(y,z)))],$$

which is generated by the execution of the G-DLS algorithm. It is easily observed that this formula has the form (5.2), where $\Phi(x,z,S)$ is $\neg(\neg R(x,z) \wedge \forall y(\neg S(x,y) \vee \neg S(y,z)))$, and $\Psi(\neg S)$ is $\top$. Thus, according to Theorem 5.2, the coherence condition for $\Theta(S)$ is $\Psi(\neg S \leftarrow \mu S(x,z).\Phi(x,z))$. Since Ψ is $\top$ and has no negative occurrences of S, its coherence condition is $\top$, which means that the required relation always exists. Moreover, we have the following explicit definition for $S(x,z)$:

$$S(x,z) \equiv \mu S(x,z).[R(x,z) \vee \exists y(S(x,y) \wedge S(y,z)].$$

7 Databases with incomplete information

So far, we have assumed that databases only contain complete information. In many AI applications, one is often confronted with the problem of using incomplete information. This could be due to the fact that we simply lack information about the properties and relations of certain objects, or that we purposely represent information incompletely for reasons of efficiency. In this section, we show how one can extend the current querying method when applied to incomplete relational databases. In order to simplify matters, we will deal with what is arguably the simplest and most naive model of incomplete information. We will only be concerned with providing a proof of concept and deal with a particular class of queries, leaving precise characterizations and variations of the approach for future research.

For simplicity, we will assume that there is a special domain value $*$, denoting the undefined value. For example, if the database contains two pairs,

$$\{(mary, smith), (john, *)\},$$

then the first pair represents $mary\ smith$ and the second pair represents a $john$ whose family name is unknown. We will also assume that if a relation contains undefined values then it cannot be directly represented in a query. Instead, we introduce new relation symbols corresponding to its 'columns'. We also assume that if a 'column' can contain undefined fields, then queries are either positive or negative w.r.t. the relation symbol corresponding to the column. This is a technical assumption that allows us to proceed without any further complications, but in certain cases can be relaxed.

Of course, since we are now dealing with incomplete information, we have to make a choice regarding the semantics of partially defined predicates. We will base this choice on intuitions from AI, where second-order circumscription is used to reason about incomplete information. More precisely, we will capitalize on the partitioning of predicates in a circumscription policy into those that are minimized, fixed, or varied. When minimizing an answer (or maximizing its negation) we will allow some partially defined predicates to vary and leave some others fixed. As in the case of circumscription, the choice of varied predicates is not immediate. A suitable heuristic might be that the varied predicates are those that are in some way related to those being minimized.

Let us assume that we are given an implicit query $\Theta(Q)$. In order to calculate a coherence condition and an explicit definition of Q we proceed as before. What will differ in this case is the algorithm used for calculating the value of the coherence condition and the relation in question. The difficulties occur when we have to calculate a value of a relation that is undefined. In such a case, we will use the following policy:

- If a relation symbol occurs positively in the query and the relation is allowed to vary then we assume $\top$ as its value.
- If a relation symbol occurs negatively in the query and the relation is allowed to vary then we assume $\bot$ as its value.
- If a relation is undefined for some object and the relation is fixed then we assume that the whole query is undefined for the object.

This solution precisely reflects the circumscription principle, where the possibly undefined predicates are allowed to vary in order to minimize a predicate. The justification and correctness of this policy easily follows from the fact that both the explicit definition and the coherence condition are monotone w.r.t. predicates occurring only positively and anti-monotone w.r.t. predicates occurring only negatively.

EXAMPLE 7.1
Assume we have a database with one binary relation $r \subseteq Name \times \{\top, \bot\} \times \{\top, \bot, *\}$, where $Name$ is a set of names of some objects. The second column indicates whether an object is a bird and the third column indicates whether it flies. For example, the contents of the database might be the following:

Name	Bird	Flies
swallow	$\top$	$\top$
Tweety	$\top$	$*$
Clyde	$*$	$*$
Leo	$*$	$\bot$

We now assume that predicate symbols B and F correspond to the last two columns of the relation. Consider the following query $\Theta(Ab)$:

$$\forall x((B(x) \wedge \neg Ab(x)) \supset F(x)),$$

where Ab which stands for *abnormal* is the intensional predicate whose minimal definition we would like to compute. According to Theorem 5.3, the coherence condition for this query would be $\top$ and the explicit definition for Ab would be $Ab(x) \equiv (B(x) \wedge \neg F(x))$, using the same method as we used for complete databases. Note that the relation F occurs positively in the query $\Theta(Ab)$. The calculated value of Ab would be $\emptyset$ independently of the choice of predicates we choose to vary.

Now consider the query $\Theta'(N)$:

$$\forall x((B(x) \wedge N(x)) \supset F(x)),$$

where N which stands for *normal* is the intensional predicate whose *maximal* definition we would like to compute. The coherence condition for this query would again be $\top$ and the explicit definition for N would be $N(x) \equiv (\neg B(x) \vee F(x))$. The following table then summarizes the possible results of calculating N.

B	F	Calculated value of N
varied	varied	$\{(swallow, \top, \top), (Tweety, \top, *), (Clyde, *, *), (Leo, *, \bot)\}$
varied	fixed	$\{(swallow, \top, \top), (Leo, *, \bot)\}$
fixed	varied	$\{(swallow, \top, \top), (Tweety, \top, *)\}$
fixed	fixed	$\{(swallow, \top, \top)\}$

It is important to note that since only the fourth step of the method for querying databases with complete information described in Section 5 differs from the querying method for querying databases with incomplete information described here, that the complexity results for steps one to three still hold. In addition, it is easily observed that step four for the new method does not add any new complexity to the querying method, so the complexity results apply to both querying databases with complete and incomplete information.

8 Conclusions

We have introduced a new declarative query language SHQL which we claim to be expressive, efficient, and natural to use. A PTIME querying method has been provided which is based on the use of quantifier elimination techniques. In addition, we have shown that the querying method may be used for databases with both complete and incomplete information. We believe the declarative character and naturalness of the query language has much to offer in comparison to logic programming approaches. SHQL is limited to only the class of PTIME queries. We are currently investigating extensions to the language and their characterizations. As stated previously, because the approach is based on an existing algorithm (DLS*), there are already cases where queries outside the class we have investigated can be *compiled* and computed. We would also like to apply these techniques to commercial relational databases and are currently working on compilation methods from SHQL into standard SQL and extended SQL.

Acknowledgements

Patrick Doherty is supported in part by the Swedish Research Council for Engineering Sciences (TFR) and the Wallenberg Foundation. Witold Łukaszewicz and Andrzej Szałas are supported in part by the State Committee for Scientific Research (Poland), KBN grant 8T11C04010.

References

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

[2] W. Ackermann. Untersuchungen über das eliminationsproblem der mathematischen logik. *Mathematische Annalen*, **110**, 390–413, 1935.

[3] A. K. Chandra. Theory of database queries. In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, pp. 1–9, 1988.

[4] P. Doherty, W. Łukaszewicz and A. Szałas. General domain circumscription and its first-order reduction. In *Proceedings of the 1st International Conference on Formal and Applied Practical Reasoning (FAPR-96)*, volume 1085 of *LNAI*, D. Gabbay and H. J. Ohlbach, eds. pp. 93–109. Springer-Verlag, Berlin, 1996.

[5] P. Doherty, W. Łukaszewicz and A. Szałas. A reduction result for circumscribed semi-horn formulas. *Fundamenta Informaticae*, **28**, 261–272, 1996.

[6] P. Doherty, W. Łukaszewicz and A. Szałas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, **18**, 297–336, 1997.

[7] P. Doherty, W. Łukaszewicz and A. Szałas. General domain circumscription and its effective reductions. *Fundamenta Informaticae*, **34**, 1–33, 1998.

[8] Y. Gurevich and S. Shelah. Fixpoint extensions of first-order logic. *Annals of Pure and Applied Logic*, **32**, 265–280, 1986.

[9] N. Immerman. Descriptive and computational complexity. In *Proceedings of Symposia in Applied Mathematics*, Volume 38, pp. 75–91, 1989.

[10] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, **68**, 86–104, 1989.

[11] A. Nonnengart and A. Szałas. A fixpoint approach to second-order quantifier elimination with applications to correspondence theory. In *Logic at Work. Essays Dedicated to the Memory of Helena Rasiowa*, E. Orlowska, ed. pp. 89–108. Physica Verlag, 1998. Also in: Report of Max-Planck-Institut für Informatik, MPI-I-95-2-007, Saarbrücken, Germany.

[12] R. Reiter. On closed world databases. In *Logic and Databases*, H. Gallaire and J. Minker, eds. pp. 55–76. Plenum Press, New York, 1978.

[13] V. Sazonov. A logical approach to the problem of P=NP. Volume 88 of *Lecture Notes in Computer Science*, pp. 562–572. Springer-Verlag, Berlin, 1980.

[14] A. Szałas. On the correspondence between modal and classical logic: An automated approach. *Journal of Logic and Computation*, **3**, 605–620, 1993.

[15] M. Y. Vardi. The complexity of relational query languages. In *14th ACM Symposium on the Theory of Computing*, pp. 137–146, 1982.

Appendix

A The DLS and G-DLS algorithm

In the Appendix we describe the DLS algorithm.[7] We also indicate all the adjustments necessary to obtain its fixpoint generalization G-DLS. The adjustments will be indicated by using the `typewriter font`. We also assume that in case of G-DLS we apply the Fixpoint Theorem instead of Ackermann's Lemma whenever the latter is not applicable. The combination of the DLS and G-DLS algorithms is referred to as DLS* in the paper.

The algorithm takes a formula of the form $\exists\Phi A$, where A is a first-order formula, as an input and returns its first-order `or fixpoint` equivalent or reports failure.[8] Of course, the algorithm can also be used for formulas of the form $\forall\Phi.A$, since the latter formula is equivalent to $\neg\exists\Phi\neg A$. Thus, by repeating the algorithm one can deal with formulas containing arbitrarily many second-order quantifiers.

The algorithm consists of four basic phases: (1) preprocessing; (2) preparation for Ackermann's Lemma; (3) application of Ackermann's Lemma; and (4) simplification. These phases are described below. It is always assumed that whenever the goal specific for a current phase is reached, then the remaining steps of the phase are skipped.

B Preprocessing

The purpose of this phase is to transform the formula $\exists\Phi.A$ into a form that separates positive and negative occurrences of the quantified predicate variable Φ. The form we want to obtain is

$$\exists\bar{x}\exists\Phi[(A_1(\Phi)\wedge B_1(\Phi))\vee\cdots\vee(A_n(\Phi)\wedge B_n(\Phi))], \tag{B.1}$$

where, for each $1 \leq i \leq n$, $A_i(\Phi)$ is positive w.r.t. Φ and $B_i(\Phi)$ is negative w.r.t. Φ.[9] It should be emphasized that not every formula is reducible into this form. `If the form (B.1) cannot be obtained we cannot proceed with the DLS algorithm, but use G-DLS instead.`

To achieve the goal of this phase, apply the steps below in the following order.

1. Eliminate the connectives $\supset$ and $\equiv$ using the usual definitions. Remove redundant quantifiers. Rename individual variables until all quantified variables are different and no variable occurs both bound and free. Using the usual equivalences, move the negation connective to the right until all its occurrences immediately proceed atomic formulas.
2. Move universal quantifiers to the right and existential quantifiers to the left applying as long as possible the following equivalences (below $Q \in \{\forall,\exists\}$, $\circ \in \{\vee,\wedge\}$ and B contains no occurrences of variables $\bar{x}$):
   - $Q\bar{x}(A(\bar{x})\circ B) \equiv (Q\bar{x}A(\bar{x}))\circ B$;
   - $Q\bar{x}(B\circ A(\bar{x})) \equiv B\circ Q\bar{x}A(\bar{x})$.
3. Move to the right the existential quantifiers that are in the scope of universal quantifiers using the equivalences of step 2.
4. Repeat (2) and (3) as long as no new existentially quantified variable can be moved into the prefix.
5. In the matrix of the formula obtained so far, distribute all top-level conjunctions over the disjunctions, containing both positive and negative occurrences of Φ, that occur among their conjuncts. For this purpose, apply the following equivalences:
   - $A\wedge(B\vee C) \equiv (A\wedge B)\vee(A\wedge C)$,
   - $(A\vee B)\wedge C \equiv (A\wedge C)\vee(B\wedge C)$,

   only if $B\vee C$ ($A\vee B$) have both positive and negative occurrences of Φ.

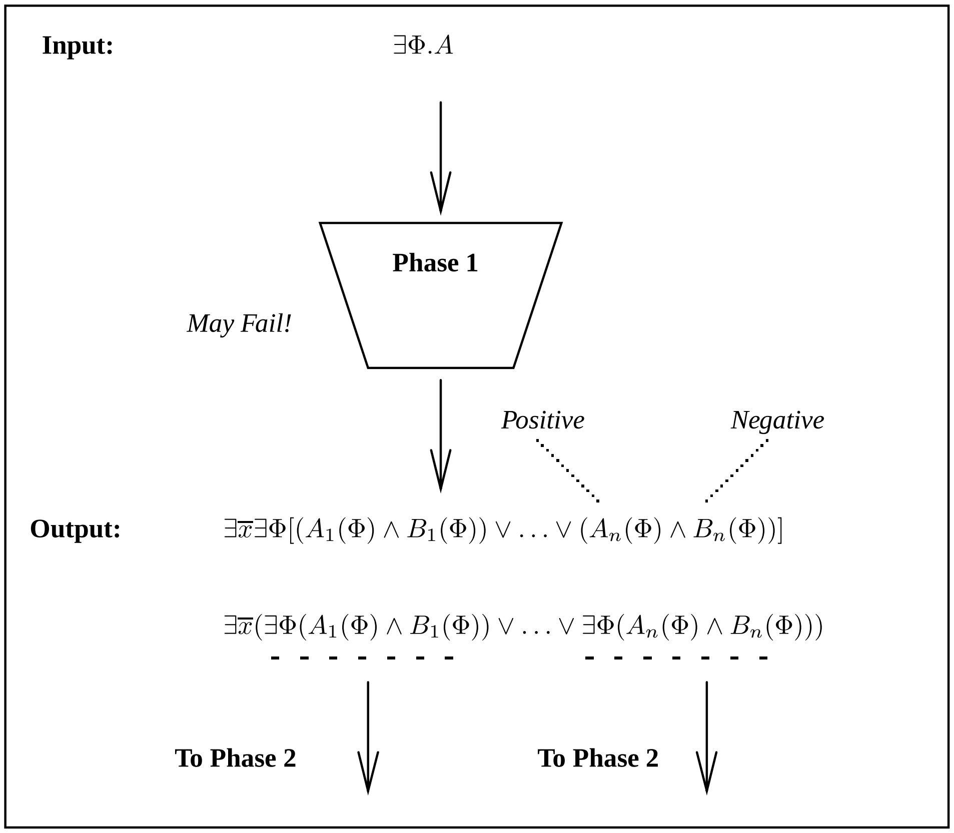
   If the resulting formula is not in the form (B.1), then report the failure of the algorithm. Otherwise replace (B.1) by its equivalent given by

$$\exists\bar{x}(\exists\Phi(A_1(\Phi)\wedge B_1(\Phi))\vee\cdots\vee\exists\Phi.(A_n(\Phi)\wedge B_n(\Phi))). \tag{B.2}$$

[7] An online implementation of the DLS algorithm may be found at the site: `http://www.ida.liu.se/labs/kplab/projects/dls/` The online version of the DLS* algorithm is currently under construction and will eventually be found at the same site.

[8] The failure of the algorithm does not mean that the second-order formula at hand cannot be reduced to its first-order `or fixpoint` equivalent. The problem we are dealing with is not even partially decidable, for first-order and `fixpoint` definability of the formulas we consider is not an arithmetical notion.

[9] To increase the strength of the algorithm, it is essential to move as many existentially quantified variables as possible into the prefix of (B.1).



Input:
$\exists\Phi.A$
Phase 1
May Fail!
Positive
Negative
Output:
$\exists\overline{x}\exists\Phi[(A_1(\Phi) \wedge B_1(\Phi)) \vee \ldots \vee (A_n(\Phi) \wedge B_n(\Phi))]$
$\exists\overline{x}(\exists\Phi(A_1(\Phi) \wedge B_1(\Phi)) \vee \ldots \vee \exists\Phi(A_n(\Phi) \wedge B_n(\Phi)))$
To Phase 2
To Phase 2

FIGURE 1. Phase 1: preprocessing the input

`In the case of the G-DLS algorithm we do not require the form (B.1).`
`Consequently, the above replacement is always done.`
For each disjunct $\exists\Phi(A_i(\Phi)\wedge B_i(\Phi))$ of (B.2) try to find its first-order `or fixpoint` equivalent by apply the next phases to the formula $\exists\Phi(A_i(\Phi)\wedge B_i(\Phi))$. If all the equivalents are obtained, return their disjunction, preceded by the prefix $\exists\bar{x}$, as the output of the algorithm.
The following example illustrates the described phase.

EXAMPLE B.1
Consider the formula

$$\exists\Phi[\forall x\exists y(P(y)\vee\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\exists z\Phi(z)\wedge\exists u\neg\Phi(u)].$$

The following lines show the subsequent transformations.

$$\begin{array}{lll}
\exists\Phi[\forall x\exists y(P(y)\vee\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\exists z\Phi(z)\wedge\exists u\neg\Phi(u)] & \equiv & \text{(by 2)} \\
\exists zu\exists\Phi[\forall x\exists y(P(y)\vee\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\Phi(z)\wedge\neg\Phi(u)] & \equiv & \text{(by 3)} \\
\exists zu\exists\Phi[\forall x(\exists yP(y)\vee\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\Phi(z)\wedge\neg\Phi(u)] & \equiv & \text{(by 2)} \\
\exists zu\exists\Phi[(\exists yP(y)\vee\forall x\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\Phi(z)\wedge\neg\Phi(u)] & \equiv & \text{(by 2)} \\
\exists zuy\exists\Phi[(P(y)\vee\forall x\exists t(\Phi(t)\vee P(x)\vee R(x,t)))\wedge\Phi(z)\wedge\neg\Phi(u)] & \equiv & \text{(by 5)} \\
\exists zuy\exists\Phi[(P(y)\wedge\Phi(z)\wedge\neg\Phi(u))\vee(\forall x\exists t(\Phi(t)\vee P(x)\vee R(x,t))\wedge & & \\
\Phi(z)\wedge\neg\Phi(u))]. & &
\end{array}$$

C Preparation for Ackermann's Lemma

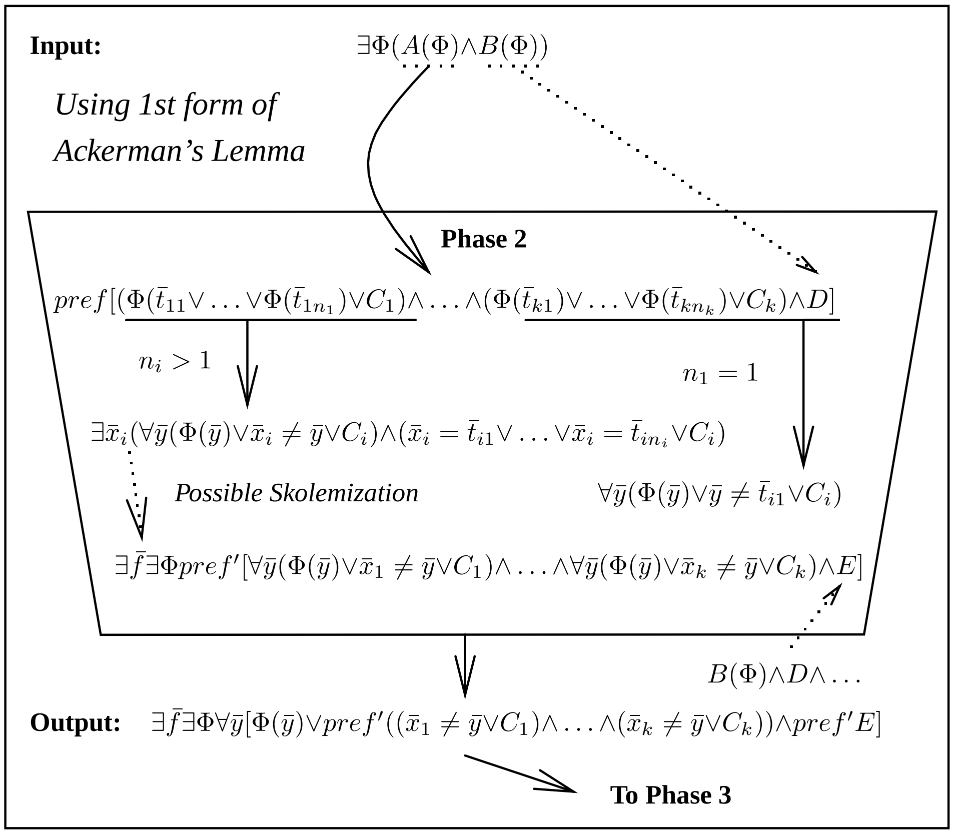

Input:
$\exists\Phi(A(\Phi)\wedge B(\Phi))$
Using 1st form of Ackerman's Lemma
Phase 2
$pref[(\Phi(\bar{t}_{11}\vee\ldots\vee\Phi(\bar{t}_{1n_1})\vee C_1)\wedge\ldots\wedge(\Phi(\bar{t}_{k1})\vee\ldots\vee\Phi(\bar{t}_{kn_k})\vee C_k)\wedge D]$
$n_i > 1$
$n_1 = 1$
$\exists\bar{x}_i(\forall\bar{y}(\Phi(\bar{y})\vee\bar{x}_i\neq\bar{y}\vee C_i)\wedge(\bar{x}_i=\bar{t}_{i1}\vee\ldots\vee\bar{x}_i=\bar{t}_{in_i}\vee C_i)$
Possible Skolemization
$\forall\bar{y}(\Phi(\bar{y})\vee\bar{y}\neq\bar{t}_{i1}\vee C_i)$
$\exists\bar{f}\exists\Phi pref'[\forall\bar{y}(\Phi(\bar{y})\vee\bar{x}_1\neq\bar{y}\vee C_1)\wedge\ldots\wedge\forall\bar{y}(\Phi(\bar{y})\vee\bar{x}_k\neq\bar{y}\vee C_k)\wedge E]$
$B(\Phi)\wedge D\wedge\ldots$
Output: $\exists\bar{f}\exists\Phi\forall\bar{y}[\Phi(\bar{y})\vee pref'((\bar{x}_1\neq\bar{y}\vee C_1)\wedge\ldots\wedge(\bar{x}_k\neq\bar{y}\vee C_k))\wedge pref'E]$
To Phase 3

FIGURE 2. Phase 2: preparation for Ackermann's Lemma

The goal of this phase is to transform a formula of the form $\exists\Phi(A(\Phi)\wedge B(\Phi))$, where $A(\Phi)$ (resp. $B(\Phi)$) is positive (resp. negative) w.r.t. Φ, into one of the forms (4.1) or (4.2) given in Lemma 4.1 or into one of the forms (4.3) or (4.4) given in Theorem 4.2. All the forms can always be obtained. However, Skolemization is sometimes necessary and unskolemization, which is to be performed in the next phase, may fail. Accordingly, the algorithm performs both transformations. Due to the symmetry of Ackermann's Lemma, the steps stated below describe only one of those transformations, namely that leading to the form (4.1) or to the form (4.3). Note that the steps stated below lead either to the form (4.1) or, otherwise, to the form (4.3).

1. Transform $A(\Phi)$ into the form

$$pref[(\Phi(\bar{t}_{11})\vee\cdots\vee\Phi(\bar{t}_{1n_1})\vee C_1)\wedge\cdots\wedge(\Phi(\bar{t}_{k1})\vee\cdots\vee\Phi(\bar{t}_{kn_k})\vee C_k)\wedge D],$$

   where $pref$ is a prefix of first-order quantifiers and Φ does not occur in $C_1,\ldots,C_k,D$. This step is always possible by applying the usual technique of obtaining the conjunctive normal form.

2. Transform each conjunct in Step 1 of form $(\Phi(\bar{t}_{i1})\vee\cdots\vee\Phi(\bar{t}_{in_i})\vee C_i)$, where $n_i>1$, into its equivalent

$$\exists\bar{x}_i(\forall\bar{y}(\Phi(\bar{y})\vee\bar{x}_i\neq\bar{y}\vee C_i)\wedge(\bar{x}_i=\bar{t}_{i1}\vee\cdots\vee\bar{x}_i=\bar{t}_{in_i}\vee C_i))$$

and move all existential quantifiers into the prefix $pref$ in Step 1. In addition, move each of the second conjuncts, $(\bar{x}_i = \bar{t}_{i1} \vee \cdots \vee \bar{x}_i = \bar{t}_{in_i} \vee C_i)$, into D in Step 1, renaming it D'.

3. Transform each conjunct in Step 1 of form $(\Phi(\bar{t}_{i1}) \vee C_i)$ into its equivalent $\forall \bar{y}(\Phi(\bar{y}) \vee \bar{y} \neq \bar{t}_{i1} \vee C_i)$.
4. Remove all existential quantifiers from the prefix $pref$ using the equivalence of Skolem given by

$$\forall \bar{x}.\exists y.A(\bar{x}, y, \ldots) \equiv \exists f.\forall \bar{x}.A(\bar{x}, y \leftarrow f(\bar{x}), \ldots), \tag{C.1}$$

where f is a new function variable. After this transformation the input formula takes the form

$$\exists \bar{f} \exists \Phi pref'[\forall \bar{y}(\Phi(\bar{y}) \vee \bar{x}_1 \neq \bar{y} \vee C_1) \wedge \cdots \wedge \forall \bar{y}(\Phi(\bar{y}) \vee \bar{x}_k \neq \bar{y} \vee C_k) \wedge E], \tag{C.2}$$

where $\bar{f}$ is the tuple of the introduced Skolem functions, $pref'$ only contains universal quantifiers, and E is $D' \wedge B(\Phi)$.

5. Transform (C.2) into its equivalent given by

$$\exists \bar{f} \exists \Phi \forall \bar{y}[\Phi(\bar{y}) \vee pref'((\bar{x}_1 \neq \bar{y} \vee C_1) \wedge \cdots \wedge (\bar{x}_k \neq \bar{y} \vee C_k)) \wedge pref' E. \tag{C.3}$$

EXAMPLE C.1 (continued)
There are two formulas to be considered in this phase, namely
$\exists \Phi(P(y) \wedge \Phi(z) \wedge \neg\Phi(u))$ and $\exists \Phi \forall x \exists t(\Phi(t) \vee P(x)) \wedge \Phi(z) \wedge \neg\Phi(u)$. We apply phase 2 to the former of the above formulas.

$$\begin{array}{lll} \exists \Phi(P(y) \wedge \Phi(z) \wedge \neg\Phi(u)) & \equiv & \text{(by 2)} \\ \exists \Phi(P(y) \wedge \forall r(\Phi(r) \vee z \neq r) \wedge \neg\Phi(u)). & & \end{array}$$

Applying phase 2 to the second formula proceeds as follows.

$$\begin{array}{lll} \exists \Phi \forall x \exists t(\Phi(t) \vee P(x) \vee R(x,t)) \wedge \Phi(z) \wedge \neg\Phi(u) & \equiv & \text{(by 3)} \\ \exists \Phi \forall x \exists t \forall r(\Phi(r) \vee r \neq t \vee P(x) \vee R(x,t)) \wedge \forall r(\Phi(r) \vee z \neq r) \wedge \neg\Phi(u) & \equiv & \text{(by 4)} \\ \exists f \exists \Phi \forall x \forall r(\Phi(r) \vee r \neq f(x) \vee P(x) \vee R(x, f(x))) & & \\ \wedge \forall r(\Phi(r) \vee z \neq r) \wedge \neg\Phi(u) & \equiv & \text{(by 5)} \\ \exists f \exists \Phi \forall r[\Phi(r) \vee (\forall x(r \neq f(x) \vee P(x) \vee R(x, f(x))) \wedge z \neq r)] \wedge \neg\Phi(u). & & \end{array}$$

D Application of Ackermann's Lemma

The goal of this phase is to eliminate the second-order quantification over Φ, applying Ackermann's Lemma, and then to unskolemize the introduced function variables. The phase consists of the following two steps.

1. Applying Ackermann's Lemma to the formula (C.3). The resulting formula is of the form

$$\exists \bar{f}[pref' E(\neg\Phi \leftarrow pref'((\bar{x}_1 \neq \bar{y} \vee C_1) \wedge \cdots \wedge (\bar{x}_k \neq \bar{y} \vee C_k)))].$$

`In the case of G-DLS the resulting formula is of the form`

$$\exists \bar{f}[pref' E(\neg\Phi \leftarrow \nu\neg\Phi(\bar{y}).pref'((\bar{x}_1 \neq \bar{y} \vee C_1) \wedge \cdots \wedge (\bar{x}_k \neq \bar{y} \vee C_k)))].$$

2. Try to remove all existential quantifiers over function variables using the equivalence (C.1). If this is impossible, the algorithm fails for the first form of Ackermann's Lemma. Using the second form returned from C, try to remove the existential quantifiers over function variables. If this is successful, go to the next step. If not, the algorithm fails.

EXAMPLE D.1 (continued)
We apply phase 3 for the pair of formulas obtained as the result of phase 2.

$$\begin{array}{lll} \exists \Phi(P(y) \wedge \forall r(\Phi(r) \vee z \neq r) \wedge \neg\Phi(u)) & \equiv & \text{(by 1)} \\ P(y) \wedge z \neq u. & & \end{array}$$

$$\begin{array}{lll} \exists f \exists \Phi \forall r[\Phi(r) \vee (\forall x(r \neq f(x) \vee P(x) \vee R(x, f(x))) \wedge z \neq r)] \wedge \neg\Phi(u) & \equiv & \text{(by 1)} \\ \exists f \forall x(u \neq f(x) \vee P(x) \vee R(x, f(x))) \wedge z \neq u & \equiv & \text{(by 2)} \\ \forall x \exists t(u \neq t \vee P(x) \vee R(x,t)) \wedge z \neq u. & & \end{array}$$

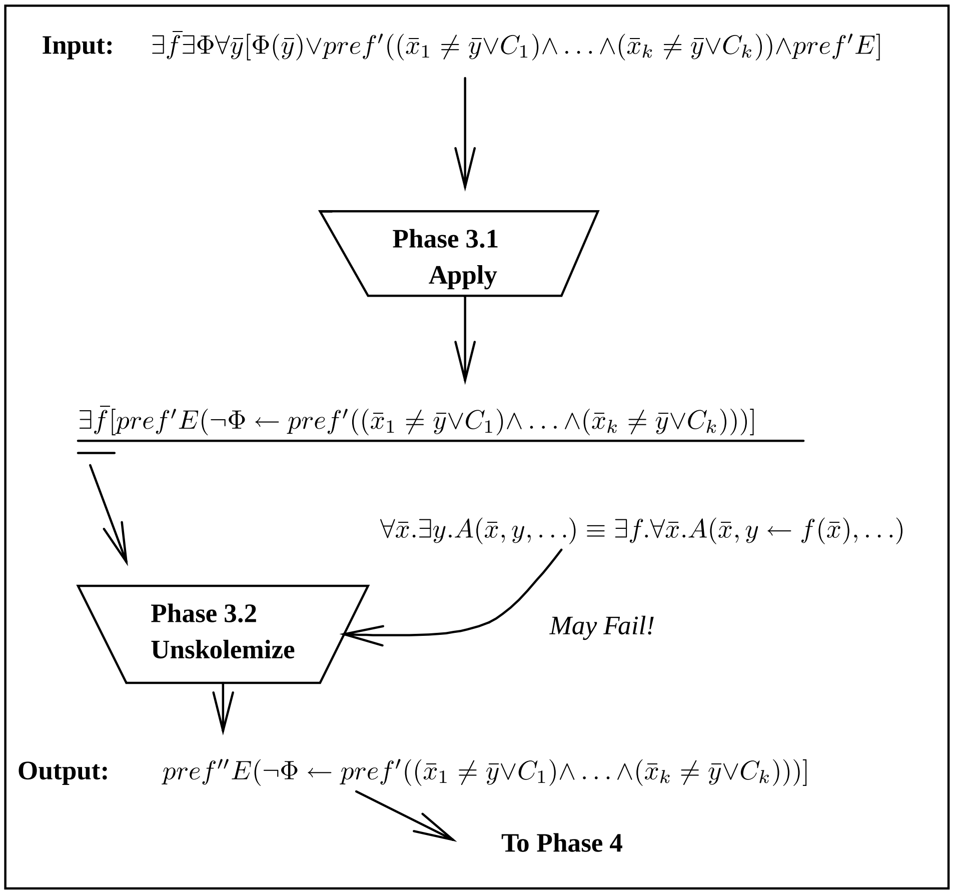

Input: $\exists\bar{f}\exists\Phi\forall\bar{y}[\Phi(\bar{y})\vee pref'((\bar{x}_1\neq\bar{y}\vee C_1)\wedge\ldots\wedge(\bar{x}_k\neq\bar{y}\vee C_k))\wedge pref'E]$
Phase 3.1 Apply
$\exists\bar{f}[pref'E(\neg\Phi\leftarrow pref'((\bar{x}_1\neq\bar{y}\vee C_1)\wedge\ldots\wedge(\bar{x}_k\neq\bar{y}\vee C_k)))]$
$\forall\bar{x}.\exists y.A(\bar{x},y,\ldots)\equiv\exists f.\forall\bar{x}.A(\bar{x},y\leftarrow f(\bar{x}),\ldots)$
May Fail!
Phase 3.2 Unskolemize
Output: $pref''E(\neg\Phi\leftarrow pref'((\bar{x}_1\neq\bar{y}\vee C_1)\wedge\ldots\wedge(\bar{x}_k\neq\bar{y}\vee C_k)))]$
To Phase 4

FIGURE 3. Phase 3: Application of Ackermann's Lemma

E Simplification

The formula obtained as the result of the previous phase can often be substantially simplified. The simplification phase consists of one step. In the formula obtained after successfully performing phase 3,

1. replace each subformula of the form $\forall\bar{x}(A(\bar{t}\leftarrow\bar{x})\vee\bar{x}\neq\bar{t})$ by $A(\bar{t})$, and
2. replace each subformula of the form $\forall\bar{x}(\bar{x}\neq\bar{t}_1\wedge\cdots\wedge\bar{x}\neq\bar{t}_n)\vee A(\bar{t}_1\leftarrow\bar{x}))$ by $A(\bar{t}_1)\wedge\cdots\wedge A(\bar{t}_n)$.

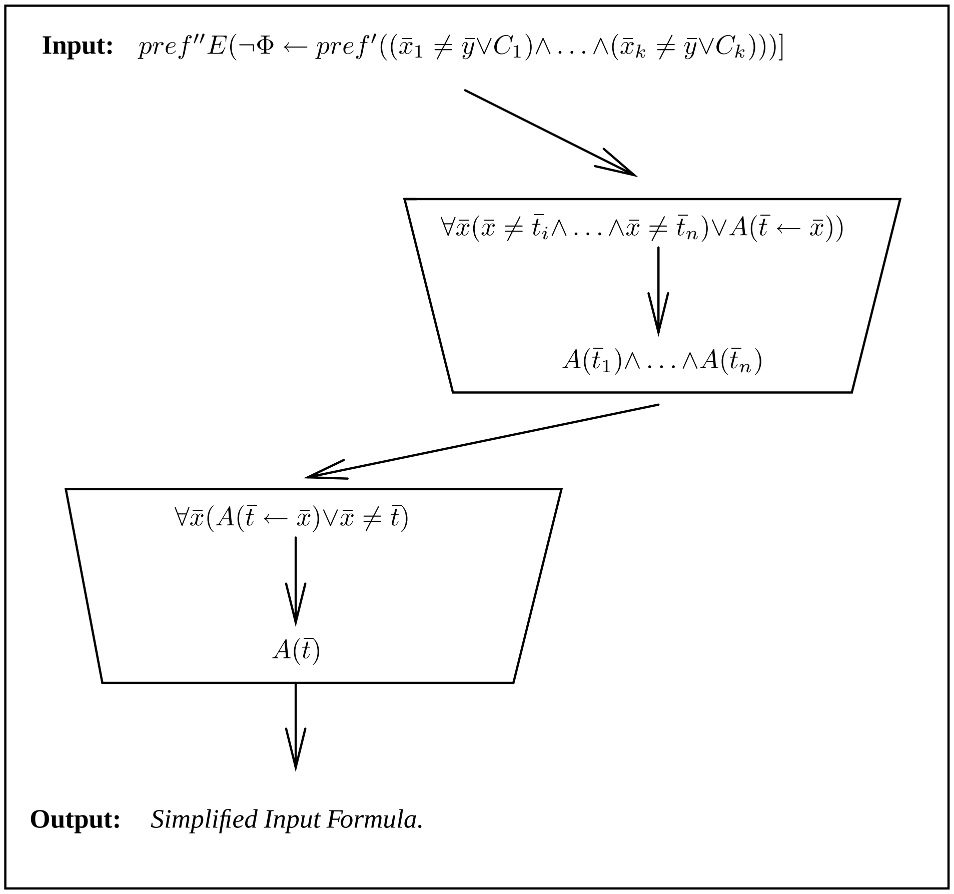

Input: $pref''E(\neg\Phi \leftarrow pref'((\bar{x}_1 \neq \bar{y} \vee C_1) \wedge \ldots \wedge (\bar{x}_k \neq \bar{y} \vee C_k)))]$
$\forall\bar{x}(\bar{x} \neq \bar{t}_i \wedge \ldots \wedge \bar{x} \neq \bar{t}_n) \vee A(\bar{t} \leftarrow \bar{x}))$
$A(\bar{t}_1) \wedge \ldots \wedge A(\bar{t}_n)$
$\forall\bar{x}(A(\bar{t} \leftarrow \bar{x}) \vee \bar{x} \neq \bar{t})$
$A(\bar{t})$
Output: *Simplified Input Formula.*

FIGURE 4. Phase 4: simplification

EXAMPLE E.1 (continued)
Since the simplification phase is inapplicable to the formulas obtained in phase 3, the first-order equivalent of the input formula we finally obtain is

$$\exists z \exists u \exists y[(P(y) \wedge z \neq u) \vee (\forall x \exists t(u \neq t \vee P(x) \vee R(x,t)) \wedge z \neq u)].$$

Received 28 October 1997