*Fundamenta Informaticae 40 (1999) 1–14*
*IOS Press*

# Meta-Queries on Deductive Databases

**Patrick Doherty**[*]
*Department of Computer and Information Science,*
*Linköping University, S-581 83 Linköping, Sweden,*
e-mail:patdo@ida.liu.se

**Jarosław Kachniarz**[†]
*Soft Computer Consultants,*
*34350 US 19N, Palm Harbor, FL 34684, USA,*
e-mail:jk@softcomputer.com

**Andrzej Szałas**[‡]
*Institute of Informatics,*
*Warsaw University,*
*ul. Banacha 2, 02-097 Warsaw, Poland,*
e-mail:szalas@mimuw.edu.pl

**Abstract.** We introduce the notion of a meta-query on relational databases and a technique which can be used to represent and solve a number of interesting problems from the area of knowledge representation using logic. The technique is based on the use of quantifier elimination and may also be used to query relational databases using a declarative query language called SHQL (Semi-Horn Query Language), introduced in [6]. SHQL is a fragment of classical first-order predicate logic and allows us to define a query without supplying its explicit definition. All SHQL queries to the database can be processed in polynomial time (both on the size of the input query and the size of the database). We demonstrate the use of the technique in problem solving by structuring logical puzzles from the Knights and Knaves

[*]Supported in part by the Swedish Council for Engineering Sciences (TFR) and the Wallenberg Foundation. Address for correspondence: Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden

[†]Address for correspondence: Soft Computer Consultants, 34350 US 19N, Palm Harbor, FL 34684, USA

[‡]Supported in part by the Wallenberg Foundation.
Address for correspondence: Institute of Informatics, Warsaw University, ul. Banacha 2, 02-097 Warsaw, Poland

domain as SHQL meta-queries on relational databases. We also provide additional examples demonstrating the flexibility of the technique. We conclude with a description of a newly developed software tool, *The Logic Engineer*, which aids in the description of algorithms using transformation and reduction techniques such as those applied in the meta-querying approach.

**Keywords:** knowledge representation, query languages, relational databases

## 1. Introduction

The knowledge representation problem in artificial intelligence is based on finding a suitable representation for the problem set to be solved and on choosing a suitable formalism for supporting the choice of representation and the reasoning required to solve problems in the set. One particularly interesting domain of problems used to test the suitability of both representations and formalisms is the domain of *logical puzzles*. Logical puzzles are quite often deceptively straightforward to state informally, yet prove to be quite difficult to represent and solve formally. Logical puzzles also provide an entertaining means of characterizing deep and subtle issues in knowledge representation. For instance, the wise man puzzle and its variants have been used by McCarthy [9] and others to study agents reasoning about the beliefs of other agents. Smullyan [13] has collected a number of puzzles to study problems related to topics such as self-reference and logical paradoxes. In particular, the solutions to a number of logical puzzles presented by Smullyan, such as some of the knights and knaves problems, are dependent on the generation of queries whose utterance contributes to the solution of the puzzles.

As an example, Smullyan [13], p. 85, describes one of the knights and knaves problems[1] as follows:

> Suppose you are an inhabitant of the island of knights and knaves. You fall in love with a girl there and wish to marry her. However, this girl has strange tastes; for some odd reason she does not wish to marry a knight; she wants to marry only a knave. But she wants a rich knave, not a poor one. (We assume for convenience that everyone there is classified as either rich or poor.) Suppose, in fact, that you are a rich knave. You are allowed to make only one statement to her. Can you convince her that you are a rich knave?

The solution to the problem is to generate a query that the rich knave should utter to the princess.

In this article, we would like to propose a novel technique for representing and solving, not only an interesting class of logical puzzles such as the one above, but other types of problems as well. The technique is based on the notion of a *meta-query* applied to a relational database. The idea is as follows. We first represent the problem to be solved as an indirect query $Q(\overline{x})$ on a relational database, where the database serves as a knowledge source or partial axiomatization

[1]On the island the knights and knaves inhabit, the knights always tell the truth and the knaves always lie.

of the problem domain. Associated with the indirect query $Q(\overline{x})$ is an additional set of logical constraints on $Q(\overline{x})$ which are part of the meta-query $\Theta(Q)$. What makes $\Theta(Q)$ a meta-query is the use of higher-order quantification where $\Theta(Q)$ is prefixed with $\exists Q$. The meta-query $\exists Q.\Theta(Q)$ asks if there exists a query $Q$ satisfying the logical constraints in $\Theta(Q)$ relative to the database.

The use of meta-queries deals with the representational aspect of the knowledge representation problem associated with this particular class of problems. What about the reasoning aspect of the knowledge representation problem? Given a representation of the problem in terms of a meta-query and a database, how do we reason about it in an efficient manner? We show that the query language SHQL, introduced in [6], can serve as an efficient language to express meta-queries. The technology developed not only provides an answer as to whether a query exists (is consistent with the database of facts), but it also produces an explicit definition for the query which can be used to compute an answer. Polynomial time complexity of both answering whether a query exists and computing an answer to the query is guaranteed. In fact, all polynomial time meta-queries are provably expressible as SHQL queries.

One of the strengths of the approach is that the representation of the problem is essentially done using a fragment of 1st-order logic to construct the meta-query. The application of the higher-order quantification is in a sense done implicitly and can in fact be viewed as a query optimization technique where the quantifier is compiled away using a quantifier elimination algorithm. On the one hand, the knowledge engineer may think about and represent a problem in terms of 1st-order logic, yet may use the power of 2nd-order logic to solve the problem implicitly in a straightforward manner.

In the case of the logical puzzles domain, this becomes important. It is often the case that, not only it is difficult to formulate the proper query to solve a logical puzzle, it is also difficult to verify whether a query actually exists to solve the puzzle. In this case, the technique we propose matches the special characteristics of the problem domain well. Queries may be formulated implicitly and one can automatically generate and verify whether a legitimate query exists relative to a particular database of facts.

Our approach is particularly interesting when compared to a number of other approaches proposed to deal with logical puzzles. Ohlbach [11], in attempting to represent the knights and knaves problem above, concludes that 1st-order logic is inadequate and that one must introduce artificial constructions into a representation of the problem in order to force a solution. Miller and Perlis [12] (also described in Thayse et al [14], pp. 46-50) pursue another route by introducing the use of indexicals in utterances and an axiomatization for utterance instances. Both solutions are dependent on a form of reification by terms, so that one may refer to utterances in the 1st-order formalization. In addition, both approaches appeal to automated theorem-proving techniques such as resolution.

Our approach avoids both the conceptual and computational complexity associated with reification and the use of resolution. Instead, we use an existential quantification over the query $Q$ and polynomial time algorithms for computing the extension of $Q$. The body of a meta-query is represented in a straightforward and direct manner using a fragment of 1st-order logic. As in Miller and Perlis [12], we can also make a claim that our technique can be generalized to solve, not only the specific logical puzzle mentioned above, but a class of logical puzzles and other classes of problems. When comparing our approach with the use of other rule-based query languages such as Datalog, we made a case for the advantages of SHQL over Datalog in Doherty et al [6]. In this paper, we show that some of the logical puzzles that are beyond the scope of Datalog are easily and naturally formulated as semi-Horn queries also introduced in Doherty et al [6].

In the rest of the paper, we will proceed as follows. In Section 2, we provide an informal description of the method. In Section 3, we demonstrate the proposed technique by formulating a variant of the knights and knaves problem described above. In addition, we derive a general query template for this particular class of problems. In Section 4, we provide the formal definitions for the query language SHQL and associated theorems used to justify the technique. In Section 5, we provide a number of additional examples where the technique is applied. Some of these examples pertain to logical puzzles and others do not. In Section 6, we consider several approaches used to implement the technique.

## 2. The Method

In this section we introduce the proposed method informally, but in more detail. The precise definitions and theorems are provided in Section 4. The proposed representation and reasoning technique allows us to ask whether there exists a desired query, satisfying certain conditions defined by the user. The conditions are formulated in the classical predicate logic, restricted to semi-Horn formulas. The query language based on semi-Horn formulas is abbreviated as SHQL (semi-Horn Query Language). In fact we will show that in certain cases one can use expressivity beyond the SHQL (see Theorem 4.3, Section 4).

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 [3]. An extension for fixpoint formulas is called the G-DLS algorithm [4, 5]. 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 always mean there is not a reduction, but simply that the algorithm can not 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 [5, 6]). 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 meta-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 meta-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. For logical puzzles such as that described in Section 1, the explicit definition of $Q$ generated in the first stage is in fact the answer to the puzzle.

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 [8], Ebbinghaus-Flum [7]).

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 e.g. Ebbinghaus-Flum [7]), our method which applies quantifier elimination techniques to semi-Horn theories makes the problem solvable in polynomial time for this special case. The coherence condition and explicit definition are also both polynomial in the size of the meta-query. Moreover, as it easily follows from a result shown in [6], all polynomial time meta-queries can be expressed in the language we deal with. Querying with SHQL is as natural as querying with classical logic and the compilation step is completely transparent to the user.

It is worth emphasizing here that we distinguish between two types of formulas: the so-called Ackermann-reducible formulas and semi-Horn formulas. In fact Ackermann-reducible formulas are also semi-Horn formulas, but allow us to calculate the coherence condition and the definition of a meta-query as a formula of the classical predicate logic. The more general case of semi-Horn formulas results in fixpoint coherence conditions and definitions of queries.

## 3. An Introductory Example

In this section, we will use a variant of the knights and knaves problem described in Section 1 to demonstrate how we can use the technique to generate an explicit query $Q$ as a solution to the problem. In addition, we describe a generic technique which allows us to solve similar problems in a straightforward manner, including the original example in Section 1.

## 3.1. Knights, Knaves and Castle Roads

Consider the following example.

**Example 3.1.** Knights, Knaves and Castle Roads.

> An island exists whose only inhabitants are knights and knaves. The knights on the island always tell the truth, while the knaves always lie. There are two roads. One of the roads leads to a castle and the other does not. An island visitor wants to ask an inhabitant of the island which road is the right road (leads to the castle), but the visitor does not know whether the person queried is a knight or a knave.

The solution to the puzzle is to find a query $Q$ which indicates the right road no matter who the visitor queries, truthful inhabitants (knights), or lying inhabitants (knaves). In the following, assume that $R(x)$ means that an inhabitant $x$ claims that a road chosen by the visitor is right (leads to the castle) and $K(x)$ means that $x$ is a knight. Since the island is only inhabited by knights and knaves, $\neg K(x)$ asserts that $x$ is a knave. We are interested in whether there is a query $Q(x)$, which when asked to an inhabitant $x$, allows us to distinguish between the right and wrong roads. If there is such a query then we should generate an explicit definition of the query. Based on the information supplied in the puzzle, query $Q(x)$ should satisfy the following conditions $\Theta(Q)$:

$$\forall x[K(x) \supset (Q(x) \equiv R(x))] \wedge [\neg K(x) \supset (Q(x) \equiv \neg R(x))]. \tag{1}$$

$\Theta(Q)$ asserts that if a knight (who always gives true answers) is queried then the query should be equivalent to asking whether the chosen road (pointed out by the visitor) is the right road, otherwise the query should be equivalent to asking whether the chosen road is the wrong road because the inhabitant queried is a knave (who always lies). Consequently, asking whether such a query exists is expressed as the following second-order formula,

$$\exists Q \forall x[K(x) \supset (Q(x) \equiv R(x))] \wedge [\neg K(x) \supset (Q(x) \equiv \neg R(x))]. \tag{2}$$

(2) is equivalent to a semi-Horn query expressed by (1). Simple transformations (that can be performed automatically) lead to the following Ackermann-reducible formula:

$$\begin{array}{l}\exists Q \forall x[\neg K(x) \vee ((\neg Q(x) \vee R(x)) \wedge (\neg R(x) \vee Q(x)))] \wedge \\ \qquad [K(x) \vee ((\neg Q(x) \vee \neg R(x)) \wedge (R(x) \vee Q(x)))],\end{array}$$

which is equivalent to,

$$\begin{array}{l}\exists Q \forall x[\neg K(x) \vee \neg Q(x) \vee R(x)] \wedge [\neg K(x) \vee Q(x) \vee \neg R(x)] \wedge \\ \qquad [K(x) \vee Q(x) \vee R(x)] \wedge [K(x) \vee \neg Q(x) \vee \neg R(x)],\end{array}$$

which is equivalent to,

$$\exists Q \forall x[Q(x) \vee (K(x) \equiv R(x))] \wedge [\neg Q(x) \vee (K(x) \equiv \neg R(x))].$$

As a result we generate the following explicit definition of $Q(x)$:

$$Q(x) \equiv (K(x) \equiv R(x)). \tag{3}$$

In addition, the coherence condition is always *true*. This simply means that the query is always consistent and gives the right answer. Translating (3) to a natural language statement with $x$ replaced by the pronoun "you" referring to the inhabitant who was queried, and "me" referring to the visitor who made the query, we would arrive at the following:

"Are you a knight if and only if the road chosen by me is the right road?"

or in other words:

"are you a knight and the road chosen by me is the right road or are you a knave and the road chosen by me is the wrong road?".

It is easy to see that this utterance stated by the visitor will provide him with the necessary information about the castle road regardless of the status of who he makes the utterance to. For example, if the visitor chooses the road to the castle and queries a knight, the knight must answer, yes (true), because he can not lie and he must evaluate the first disjunct in the query to true. Consequently, the whole query is evaluated to true. In another case, if the visitor chooses the road to the castle and queries a knave, the knave evaluates both disjuncts in the query to false, but because he must lie, he must also answer yes (not false). The remaining two cases are similar. □

## 3.2. A Generic Approach to a Class of Knights and Knaves Puzzles

Given the example above, it is easily observable that if an inhabitant $x$ of the island is asked any question of the form $R(x, \bar{y})$, where $y$ is a touple of variables, the question has to satisfy the condition (1). This proves that such a query always exists (the coherence condition is *true*) and the query is defined by (3). For example, suppose the visitor would like to know whether an inhabitant is a knight ($R(x, \overline{y})$ is $K(x)$), then the proper query would have the form,

$$Q(x) \equiv (K(x) \equiv K(x)),$$

which is equivalent to *true*. This means that we simply ask the inhabitant about any tautology. In fact, there is now a more powerful way to ask inhabitants questions if the visitor is allowed two queries. First determine if the inhabitant is a knight or a knave and then ask any question.

The example in Section 1 can now be solved in two ways, either by setting up the meta-query in the standard manner or by using the generic technique. In the first case the meta-query would be,

$$\exists Q \forall x[\neg K(x) \supset (Q(x) \equiv \neg R(x))] \wedge [K(x) \supset \neg Q(x)],$$

where $R(x)$ denotes $x$ is rich, $K(x)$ denotes $x$ is a knight and in this case $x$ denotes $I$ because an inhabitant will utter the statement to the princess. This meta-query is semi-Horn and reduces to the following explicit definition for $Q(x)$,

$$Q(x) \equiv (\neg K(x) \wedge \neg R(x)), \tag{4}$$

with coherence condition *true*. The natural language translation and solution to the puzzle is

I am not a knight and I am not rich.

In the case where the generic technique is used ($R(x, \overline{y})$ is $\neg K(x) \wedge R(x)$), (3) would have the form,

$$Q(x) \equiv (K(x) \equiv (\neg K(x) \wedge R(x))),$$

which is equivalent to (4).

## 4. The Semi-Horn Query Language

In the following, we will define the semi-Horn query language (SHQL) and then present the formal justifications used as a basis for automatically generating coherence conditions and explicit definitions for SHQL meta-queries. Note that the techniques described below can sometimes be applied successfully to meta-queries more expressive than those constructed using semi-Horn formulas.

### 4.1. Semi-Horn Formulas

We shall consider two types of formulas of the form

$$\Phi_1(Q) \wedge \Phi_2(Q), \tag{5}$$

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) for which $\Phi_1(Q)$ is a conjunction of formulas of the form $\forall \bar{x}(Q(\bar{t}) \vee \Phi)$, where $\Phi$ is an arbitrary $Q$-free first-order formula
- *semi-Horn formulas (w.r.t. $Q$)* are of the form (5) for which $\Phi_1(Q)$ is a conjunction of formulas of the form $\forall \bar{x}(Q(\bar{t}) \vee \Phi(\neg Q))$, and $\Phi$ 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 $\Phi$ 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. However, it is important to isolate this class of formulas because these define first-order expressible queries.

**Definition 4.1.** By a *declarative SHQL 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.

In what follows we concentrate on semi-Horn and Ackermann-reducible formulas. It is, however, important to note that the method can be made more general by generating solutions for arbitrary formulas , $(Q)$ which although not semi-Horn, are reducible e.g. by the DLS* algorithm described in [5].

Note that any conjunction of semi-Horn formulas w.r.t. $Q$ can be transformed into the following form (see e.g. [6]):

$$\forall \bar{x}[\Phi(\bar{x}, \overline{z_i}, Q) \supset Q(\bar{x})] \wedge \Psi(\neg Q), \tag{6}$$

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,

$$\forall \bar{x}[\Phi(\bar{x}, \overline{z_i}, \neg Q)) \supset \neg Q(\bar{x})] \wedge \Psi(Q), \tag{7}$$

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. $Q$.

## 4.2. Coherence Conditions and Explicit Definitions

The following Theorems 4.1 and 4.2 of [6], based on Ackermann's lemma [2] and Nonnengart and Szałas fixpoint theorem [10], provide us with both coherence conditions and explicit definitions for SHQL queries.

**Theorem 4.1.**
For any formula $\Theta(Q)$ of the form (6):

- 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 (7):

- 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 4.2.**
For any formula $\Theta(Q)$ of the form (6), where $\Phi$ 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 (7), where $\Phi$ 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 $\Psi(Q \leftarrow \neg\Phi(\bar{x}, \overline{z_i}))$. □

Let us also introduce a new theorem (Theorem 4.3 below) which allows us to go beyond the semi-Horn query language of [6] and avoid some superfluous transformations of the formulas to the forms considered before and allows us to deal with a more general case.

**Theorem 4.3.**
For any formula of the form: $\Theta(Q) \equiv \forall\bar{x}[\Phi(\bar{x},\overline{z_i},Q) \equiv Q(\bar{x})] \wedge \Psi(\neg Q, Q)$, where $\Psi(\neg Q, Q)$ is an arbitrary first-order formula that may contain both negative and positive occurrences of $Q$ and $\Phi(\bar{x},\overline{z_i},Q)$ is positive w.r.t. $Q$:

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

## 5. Examples

In this section, we introduce two additional examples unrelated to solving logical puzzles that demonstrate other uses of the technique.

**Example 5.1.** This example is considered in [1](Example 15.3.1, p.386) and its characterization using the technique in this paper uses fixpoint formulas.

> "The example concerns a game with states $a, b, \ldots$. The game is between two players. The possible moves of the games are held in a binary relation *moves*. A tuple $\langle a, b\rangle$ in *moves* indicates that when in a state $a$, one can chose to move to state $b$. A player loses if he or she is in a state from which there are no moves. The goal is to compute the set of winning states (i.e., the set of states such that there exists a winning strategy for a player in this state). These are obtained as the extension of a unary predicate *win*."

Let $M(x,y)$ and $W(x)$ denote the predicates *moves* and *win*, respectively. The solution proposed in [1] is formulated using the following nonstratifiable program definition:

$$W(x) \leftarrow M(x,y), \neg W(y),$$

which states that $x$ is a winning state if there is a move from $x$ to a state $y$, "for which the opposing player loses". Unfortunately, $\neg W(y)$ means that there is not a winning strategy from $y$, not that "a player loses starting from $y$" as is assumed in [1]. Moreover, a three-valued semantics associated with datalog with negation is used to compute the answers.

Let us formulate the solution correctly and show that negation can be interpreted as classical negation and that *datalog* can be replaced by the use of formulas in classical logic. In order to show that we will ask whether there is a query that allows us to calculate $W$, i.e. whether there is a $W$ satisfying the following conditions:

1. $\forall x[(\exists y M(x,y) \wedge \forall z \neg M(y,z)) \supset W(x)]$, i.e. from $x$ there is a move to $y$ from which the opposing player has no move;
2. $\forall x[(\exists y M(x,y) \wedge \forall z(M(y,z) \supset W(z))) \supset W(x)]$, i.e. from $x$ there is a move to $y$ from which all choices of the opposing player lead to a state where the other player (the player that moved from $x$ to $y$) has a winning strategy.

Combining (1) and (2) and using both in a meta-query for $W$ results in the following second-order formula:

$$\exists W \forall x[W(x) \vee \forall y \neg M(x,y) \vee \exists z M(y,z)] \wedge [W(x) \vee \forall y \neg M(x,y) \vee \exists z M(y,z) \wedge \neg W(z)],$$

which can easily be reduced to the semi-Horn formula form in (6),

$$\exists W \forall x[W(x) \vee \forall y \neg M(x,y) \vee \exists z M(y,z) \wedge \exists z' M(y,z') \wedge \neg W(z')].$$

According to 4.1 such a query always exists (the coherence condition is *true*) and its definition is given by,

$$W(x) \equiv \mu W(x).[\exists y M(x,y) \wedge (\forall z \neg M(y,z) \vee \forall z' \neg M(y,z') \vee W(z'))].$$

The computation algorithm is straightforward. In the first step,

$$W^{(1)}(x) \equiv \exists y M(x,y) \wedge \forall z \neg M(y,z),$$

mark as winning all states for which there is a move to another state but from that state there is not any move. In the next steps,

$$W^{(n)}(x) \equiv \exists y M(x,y) \wedge (\forall z \neg M(y,z) \vee \forall z' \neg M(y,z') \vee W^{(n-1)}(z'))],$$

mark as winning all states from which there is a move to another state from which there is not any move, or all moves lead to a state already identified as a winning one. □

The folowing example shows the usefulness of Theorem 4.3.

**Example 5.2.** The example of applying semi-Horn queries in a Clinical Information System.

Let us consider a Clinical Information System storing, among others, patient demographic data (including parent-child relationship) as well as diagnosed disease cases. Let the relation $P(x,y)$ mean that $x$ is a parent of $y$ and $D(x)$ mean that patient $x$ fell ill with a certain disease $d$. The researcher may want to ask a query: "Is there any data in the database confirming the hypothesis that the disease $d$ is hereditary?". He wants to compute a relation $A(x,y)$ containing all cases when both ancestor $x$ and descendant $y$ had the same illness $d$.

The query $A(x,y)$ has to satisfy the following conditions:

1. $\forall x[D(x) \supset \exists z A(z,x) \wedge D(z)]$, i.e. if $x$ had the disease, there is an ancestor of $x$ who also had this disease;
2. $\forall x \forall y[A(x,y) \equiv P(x,y) \vee \exists z P(z,y) \wedge A(x,z)]$, i.e. $x$ is an ancestor of $y$ if and only if $x$ is $y$'s parent or there is $x$'s descendant $z$, who is a parent of $y$;
3. $\forall x \forall y[A(x,y) \supset \neg A(y,x)]$, i.e. if $x$ is an ancestor of $y$, $x$ cannot be a descendant of $y$;

The equivalent second order formula has the form:

$$\exists A \forall x \forall y [A(x,y) \equiv P(x,y) \vee \exists z P(z,y) \wedge A(x,z)] \wedge \\ [D(x) \supset \exists z A(z,x) \wedge D(z)] \wedge [A(x,y) \supset \neg A(y,x)].$$

According to Theorem 4.3, the explicit definition of $A(x,y)$ is given by:

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

and the coherence condition, after simple transformations, has the form:

$$\forall x \forall y \exists t_1 \exists t_2 (\neg \mu A(t_1,t_2).[P(t_1,t_2) \vee \exists z P(z,t_2) \wedge A(t_1,z)] \wedge (t_1 = x \wedge t_2 = y \vee t_1 = y \wedge t_2 = x)) \\ \wedge \forall x (\neg D(x) \vee \exists y \mu A(y,x).[P(y,x) \vee \exists z P(z,x) \wedge A(y,z)] \wedge D(y)).$$

□

## 6. Implementation

### 6.1. Logic Engineer

There are many formula reduction algorithms, including some that eliminate second-order quantifiers such as the DLS algorithm [3, 6] used in this paper[2]. In theory, many of these algorithms can be applied without any computer aid, but in practice many of the reductions require complex and robust transformations. We learned this ourselves through the experience developing the DLS algorithm. Since a great deal of research related to the practical use of logic requires algorithms with extensive use of formula transformations and reductions, there is a need for a practical toolkit for logic engineers that would include a library of definable rules (known a priori tautologies) and reduction algorithms that would support the transformation of input formulas into a desired form. This form could range from anything as straightforward as DNF or CNF to sophisticated quantifier elimination algorithms.

An important feature of any such toolkit should be the ability to incrementally refine existing transformation and reduction techniques in a modular manner in addition to adding new techniques. The DLS algorithm is a case in point. The restriction of the answering technique to semi-Horn formulas in Doherty et al. [6] resulted from the fact that there are very simple formulas which are not semi-Horn and which result in NP-complete answering algorithms. On the other hand, as research progresses, one can discover new theorems and algorithms optimizing certain transformations of formulas into semi-Horn forms (as is done in Theorem 4.3). In this respect, *hard-coding* transformations of formulas into the DLS algorithm is not sufficient when one expects refinement of such algorithms by adding new transformation rules leading to more compact results, for example.

[2]An on-line implementation of the DLS algorithm can be accessed via `http://www.ida.liu.se/labs/kplab/project/dls/`.

The **Logic Engineer**, which is part of a larger educational project called the *Computer Aided Logic Engineering Project*, is intended to meet these expectations and requirements. It is being developed as a teaching and research aid where the user can easily define *transformation rules* and algorithms and observe their behavior in terms of the rules. It also provides the ability to verify that algorithms are correct and properly described. The Logic Engineer is packaged as a GUI application written in Java and may be run on any platform compatible with Java VM 1.1. All examples from this paper have been calculated using this system.[3]

## 6.2. Transformation Rules and Algorithm

The knowledge used by the Logic Engineer is represented as a collection of rules representing known tautologies which can be used in formula transformations. A sample rule may have the form:

$$\textbf{DEFINE RULE}\, \underbrace{NOT(a\ OR\ b)}_{pattern formula}\, \textbf{EQUIV}\, \underbrace{NOT\ a\ AND\ NOT\ b}_{equivalent formula}$$

The number of rules required for a particular problem or algorithm depends on the complexity of the problem or algorithm. For example, simple algorithms like transformation to CNF (conjunctive normal form) can be defined using ten rules, while complex algorithms such as the DLS algorithm require about fifty.

The transformation algorithm used in the Logic Engineer defines a sequence of transformations leading to a desired form of an output formula. The language used to describe algorithms, called HDL, is declarative and domain-oriented. The coded algorithm is very similar in form to human readable descriptions published in articles. It consists of a sequence of statements of the form: "while the formula is still changing, apply these rules"; "if the formula has a certain form apply these rules, otherwise use these other rules"; "apply these rules to any subformula matching the given criteria", etc. Although the Logic Engineer simplifies the implementation of reduction and transformation algorithms, it is still not trivial to provide the declarative rule-based representation of the algorithm even when the algorithm is already published. Quite often, authors of algorithms skip the "trivial" parts of the algorithm, where these parts are usually the most complex for the computer. Less complex algorithms like CNF can be coded in about 50 lines, whereas complex algorithms such as the DLS algorithm require roughly 350 lines.

## References

[1] Abiteboul, S., Hull, R., Vianu, V. (1996) *Foundations of Databases*, Addison-Wesley Pub. Co.

[2] Ackermann, W. (1935) *Untersuchungen über das Eliminationsproblem der mathematischen Logik*, Mathematische Annalen, 110, 390-413.

[3] More information about the Logic Engineer may be found at `http://zls.mimuw.edu.pl/~szalas/cale/`. It will be made available for use in Spring of 2000.

[3] Doherty, P., Łukaszewicz, W., Szałas, A. (1997) *Computing Circumscription Revisited. A Reduction Algorithm*, Journal of Automated Reasoning, 18, 3, 297–336 .

[4] Doherty, P., Łukaszewicz, W., Szałas, A. (1996) *A Reduction Result for Circumscribed Semi-Horn Formulas*, Fundamenta Informaticae, 28, 3-4, 261–272.

[5] Doherty, P., Łukaszewicz, W., Szałas, A. (1998) *General Domain Circumscription and its Effective Reductions*, Fundamenta Informaticae, 36, 1, 23–55

[6] Doherty, P., Łukaszewicz, W., Szałas, A. (1999) *Declarative PTIME Queries for Relational Databases using Quantifier Elimination*, Journal of Logic and Computation, 9, 739-761.

[7] Ebbinghaus, H.-D., Flum, J. (1995) *Finite Model Theory*, Springer-Verlag.

[8] Immerman, N. (1986) *Relational Queries Computable in Polynomial Time*, Information and Control, 68, 86-104.

[9] McCarthy, M., Sato, M., Hayashi, T., Igarashi, S. (1978) *On the Model Theory of Knowledge*, Stanford Artificial Intelligence Laboratory, Memo AIM-312, Stanford University.

[10] Nonnengart, A., Szałas, A. (1998) *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. Orłowska (ed.), Physica Verlag, 89-108.

[11] Ohlbach, H. J. (1984) *Predicate Logic Hacker Tricks*, Journal of Automated Reasoning 1, pp,435-440.

[12] Miller, M., Perlis, D. (1987) *Proving Self-Utterances*, Journal of Automated Reasoning (3)3, 329-338.

[13] Smullyan, R. (1978) *What is the Name of This Book?*, Prentice Hall, Englewood Cliffs, New Jersey.

[14] Thayse, A. (ed.) (1989) *From Modal Logic to Deductive Databases - Introducing a Logic Based Approach to Artificial Intelligence*, John Wiley & Sons.