Knowledge compilation = Query rewriting + View synthesis

Marco Cadoli, Toni Mancini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli|tmancini@dis.uniroma1.it

ABSTRACT

In Knowledge Compilation (KC) an intractable deduction problem $KB \models f$ is split into two phases: 1) KB is preprocessed, thus obtaining a data structure D_{KB}; 2) the problem is efficiently solved using D_{KB} and f. Our goal is to study KC in the context of relational databases: Both KB and f are represented as databases, and '$\models$' is represented as a query Q in second-order logic. D_{KB} is a database, to be *synthesized* from KB by means of an appropriate *view*. Q is *rewritten*, thus obtaining Q_r. We show syntactic restrictions on Q implying that a polynomial-size D_{KB} and a first-order Q_r exist, which imply that phase 2 can be done in polynomial time. We also present classes of queries (in some sense complementary to the former ones) for which either no polynomial-size D_{KB} or no first-order Q_r exist (unless the PH collapses). Compilation to other complexity classes is also addressed.

Keywords: Logic in databases, complexity, deductive databases.

1. INTRODUCTION

Knowledge compilation (KC) is one of the techniques that have been proposed in the Artificial Intelligence literature for addressing polynomial intractability of reasoning. The central observation is that, in many reasoning tasks, the input to a deduction problem $KB \models f$ is split into two parts, KB and f, with different status: Typically KB is not modified very often, and it is used for several instances of the problem. For this reason, KB is sometimes called the *fixed part*, and f the *varying part*. The idea of KC is to split the deduction problem into two phases (depicted in Figure 1):

- in the first one (sometimes called *off-line reasoning*) KB is preprocessed, thus obtaining an appropriate data structure D_{KB};

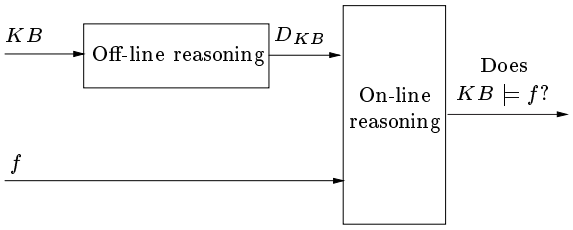

Figure 1: Compilation of a problem

- in the second phase (sometimes called *on-line reasoning*), the problem $KB \models f$ is actually solved using D_{KB} and f.

The goal of preprocessing is to make on-line reasoning computationally easier (possibly polynomial-time) with respect to reasoning in the case when no preprocessing at all is done. A reasonable requirement is that the size of D_{KB} is polynomial in the size of KB, otherwise it would be unpractical to store the computed information. Intuitively, the effort spent in the preprocessing phase pays off when its computational cost is amortized over the (facilitated) answer to many instances. KC has been analyzed for a variety of logical consequence relations $\models$, including propositional logic [18], belief revision [20], and various non-monotonic formalisms such as default logic [1] and circumscription [17].

The goal of the present paper is to study KC in the context of relational databases. In particular, we will assume that both KB and f are represented in some way as databases, and that logical consequence, i.e., the '$\models$' relation in $KB \models f$, is represented as a logical query Q in second-order (SO) logic. D_{KB} is also a database, to be *synthesized* from KB by means of an appropriate *view*. On-line reasoning is also represented as a logical query Q_r, obtained by *rewriting* Q.

We will show syntactic restrictions on the query Q which imply that a polynomial-size D_{KB} and a first-order (FO) Q_r exist, thus implying that on-line reasoning can be done in polynomial time, i.e., Q is *compilable* to P. We will also present classes (in some sense complementary to the former ones) of *non-compilable* queries, for which either no polynomial-size D_{KB} or no FO Q_r exist (unless the Polynomial Hierarchy -PH- collapses).

Focusing on such a restricted case of KC rules out some interesting cases: As an example, since Q is SO, no problem which is PSPACE-hard can be studied (provided PH $\neq$ PSPACE). Nevertheless, formalizing the deduction problem as a logical formula presents a clear advantage, since we have the possibility to separate compilable and non-compilable queries by means of a syntactic criterion. This helps to understand why "similar" problems may be quite different with respect to compilation.

In fact a vast literature both on the pragmatic and the theoretical aspects of KC exists (a survey on knowledge compilation appears as [2]). Apart from the above cited papers, many cases of compilable and non-compilable problems have been shown in [9, 6]; ad hoc complexity classes and reductions for proving compilability and non-compilability have been defined in [3]. Nevertheless the syntactic approach considered in the current paper is, to the best of our knowledge, original.

The syntactic approach has been used in other works, e.g., in [10] for characterizing polynomial and NP-complete subcases of existential SO logic over graphs, and in [13] for characterizing approximable NP optimization problems.

The paper is organized as follows: In Section 2 we recall preliminary notions and present a couple of motivating examples, while in Section 3 the main theorem concerning queries compilable to P is presented. In Section 4 we investigate the reasonableness of the assumptions of such a theorem, showing that, relaxing them, non-compilable problems can be expressed. Section 5 generalizes results of Section 3 by dealing with compilability to other complexity classes. Several examples will be described throughout the paper. Conclusions are drawn in Section 6.

2. PRELIMINARIES AND MOTIVATING EXAMPLES

In this section we give the preliminary notions that will be used throughout the paper. First of all, we give the formal definition of *problem compilable to* P; such a definition refers only to problems whose input is split into a fixed and a varying part. Afterwards, we use a couple of specific examples of deduction problems, to highlight the intuitions behind the definitions. In the following, $|\cdot|$ denotes the size of a structure.

Definition 2.1 (Problem compilable to P [4, 18]). *A problem π, which is split into a fixed and a varying part, FIX and VAR respectively, is said to be* compilable to P *if there exist two polynomials p_1, p_2 and an algorithm ASK such that for each instance KB of FIX there is a data structure D_{KB} such that:*

1. *$|D_{KB}| \leq p_1(|KB|)$;*
2. *for each instance f of VAR the call $ASK(D_{KB}, f)$ returns yes iff $\langle KB, f\rangle$ is a "yes" instance of π;*
3. *$ASK(D_{KB}, f)$ requires time $\leq p_2(|f| + |D_{KB}|)$.*

The problems we consider in this section refer to propositional logic, and in particular to deduction from a propositional formula in conjunctive normal form (CNF), i.e., a conjunction of *clauses*, where a clause is a disjunction of *literals*, i.e., propositional variables, possibly negated.

The problems we consider are conj-inf and clause-inf, whose definitions are the following:

Problem 2.1 (Conjunction inference – conj-inf).

Fixed part *Propositional formula T in CNF;*

Varying part *Conjunction of literals $J = l_1 \wedge \cdots \wedge l_n$;*

Question *Does $T \models J$, i.e., is every model of T also a model of J?*

Problem 2.2 (Clause inference – clause-inf).

Fixed part *Propositional formula T in CNF;*

Varying part *Disjunction of literals $C = l_1 \vee \cdots \vee l_n$;*

Question *Does $T \models C$, i.e., is every model of T also a model of C?*

To make above specifications more clear, consider the following formula T:

$$(\neg c \vee d) \wedge (a \vee \neg b) \wedge (c \vee \neg a) \wedge (\neg d \vee \neg a). \qquad (1)$$

Formula T logically implies both the conjunction $\neg a$ and the clause $c \vee \neg b$.

As mentioned in Section 1, in our setting the problem instance must be represented as a relational database. As for the fixed part, there are several ways to encode a formula in CNF as a database (cf. e.g., [10]). We choose to represent a formula $T = \gamma_1 \wedge \cdots \wedge \gamma_m$ over the set of variables $x_1, \ldots, x_n$ as a directed graph, i.e., a binary relation E_T, in which:

- there is one node for each clause γ_i $(1 \leq i \leq m)$, and one node for each of the literals x_i, $\neg x_i$ $(1 \leq i \leq n)$;
- for each variable x_i there is one edge from x_i to $\neg x_i$ and one edge from $\neg x_i$ to x_i;
- for each clause $\gamma_i = l_i^1 \vee \cdots \vee l_i^{k_i}$, there are k_i edges ending in γ_i, one for each literal $l_i^1, \ldots, l_i^{k_i}$.

Note that clauses correspond to sink nodes in the graph, and that complementary literals form a cycle of length 2. We call a relation like E_T a *fixed* relation.

In Figure 2 we show the graph corresponding to the CNF formula (1).

The varying parts of both problems can be simply represented as unary relations: C for the clause, and J for the conjunction, storing tuples corresponding to their literals; by referring to the previous example, conjunction $\neg a$ and the clause $c \vee \neg b$ (both of them logically implied by the CNF formula in Figure 1), are represented by the following relations:

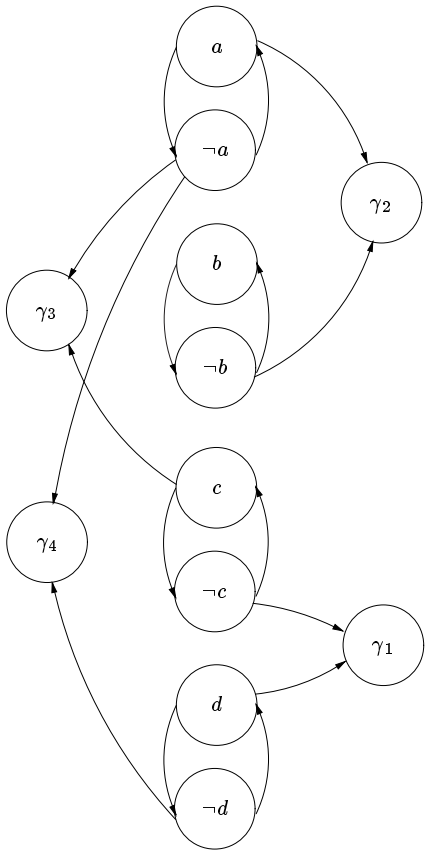

Figure 2: Graph of the CNF formula (1)

J
$\neg a$

C
c
$\neg b$

We call relations like J and C *varying* relations.

Now, we have to represent the deduction problems CONJ-INF and CLAUSE-INF as queries. Since both problems are coNP-complete, queries must be formulae of universal SO logic ($SO_\forall$), cf. [8]. Before showing the complete queries, we go through some intermediate formulae, which are useful for what follows.

A given unary relation M represents an *interpretation* of a CNF formula T, represented as relation E_T, iff the following FO formula evaluates to true:

$$\forall X\ \forall Y\ E_T(X,Y) \wedge E_T(Y,X) \rightarrow \\ ((M(X) \wedge \neg M(Y)) \vee (M(Y) \wedge \neg M(X))).$$

The above formula, which will be abbreviated as $INT(M, E_T)$, asserts that for each pair of complementary literals X and Y, exactly one is in M (implicitly getting truth value true). By referring again to formula (1), a possible interpretation assigns true to a, c, d and false to b, and it is encoded by the following relation M:

M
a
$\neg b$
c
d

Note that this relation makes the previous formula true.

Interpretation M is a *model* of T iff it assigns true to at least one literal Z for each clause X, i.e., for every sink node of E_T. Formally, M is a model of T iff the following FO formula (abbreviated as $MOD(M, E_T)$) evaluates to true:

$$INT(M, E_T) \wedge \\ \forall X\ \forall Y \neg E_T(X,Y) \rightarrow \exists Z\ (E_T(Z,X) \wedge M(Z)). \qquad (2)$$

Coming back to the example, it can be observed that interpretation M in the above table (which does not satisfy clause γ_4 in formula (1)) makes formula (2) false. On the other hand, the following one (a model of (1)) makes it true:

M
$\neg a$
$\neg b$
c
d

Problems CONJ-INF and CLAUSE-INF can now be simply represented, respectively, as the following $SO_\forall$ queries:

$$\forall M\ MOD(M, E_T) \rightarrow \forall X\ (J(X) \rightarrow M(X)) \qquad (3)$$

$$\forall M\ MOD(M, E_T) \rightarrow \exists X\ (C(X) \wedge M(X)). \qquad (4)$$

It is well-known, cf. [18], that the problem CONJ-INF can take advantage of compilation: Given a CNF formula T and a conjunction δ, off-line reasoning amounts to computing the set of literals logically following from T, i.e., $V = \{l \mid l$ is a literal and $T \models l\}$, while on-line reasoning just checks that each literal occurring in δ belongs to V. Since the set V has polynomial size in the size of T, and on-line reasoning is polynomial-time in the size of V and δ, CONJ-INF is said –according to Definition 2.1– to be compilable to P.

This idea can be readily used by transforming the $SO_\forall$ query (3) as follows. Off-line reasoning is captured by computing a *view*, i.e., the following SO formula open with respect to FO variable X, which will be abbreviated as $INFLIT(X)$:

$$\forall M \; MOD(M, E_T) \rightarrow M(X). \tag{5}$$

By referring to formula (1), since all its models (extensions for relation M that make formula (2) true) are:

$\neg a$
$\neg b$
$\neg c$
$\neg d$

$\neg a$
$\neg b$
$\neg c$
d

$\neg a$
$\neg b$
c
d

it follows that $INFLIT(X)$ would store the following literals:

$INFLIT$
$\neg a$
$\neg b$

On-line reasoning is done by evaluating the following query, obtained by *rewriting* (3):

$$\forall X \; J(X) \rightarrow INFLIT(X). \tag{6}$$

In our example, the conjunction J is $\neg a$, therefore formula (6) is true.

In general, computing the view (5) is an NP-hard task, but in KC we don't care about it: What matters is that the view needs polynomial space to be stored, and that the rewritten query (6) is FO, hence polynomial-time.

It is possible to prove that the rewritten query (6) is equivalent to the original one (3). Actually, in the next section we show a subset of SO queries – generalizing (3) – for which the existence of a polynomial-size view and a FO equivalent query is guaranteed.

The situation is radically different for the CLAUSE-INF problem, for the following reason. Given a CNF formula T and a clause γ, off-line reasoning could – in principle – be done by computing all clauses which logically follow from T, and on-line reasoning just by checking whether γ is one of them or not. More compactly, only the clauses not implied by other ones, i.e., the so called *prime implicates*, could be computed during off-line reasoning. Anyway, even the latter method is not practical, since it has been proven in [5] that there exist CNF formulae in which the number of prime implicates is exponential in their size.

The definition of non-compilable problem follows from Definition 2.1: π is *non-compilable to* P if there are no polynomials p_1, p_2 with the properties stated in Definition 2.1. A necessary (but not sufficient) condition for non-compilability is that the set VAR of possible varying parts of the input is super-polynomial in the size $|KB|$ of the fixed part. Intuitively, if a polynomial p_1 with the properties mentioned in Definition 2.1 does not exist, it means that compilation would take too much (i.e., super-polynomial) space. Dually, if p_1 exists and p_2 does not, then encoding of the problem in a polynomial-space data structure is possible, but it is not possible to extract information from it in polynomial time.

Actually, it is still not known whether the CLAUSE-INF problem is compilable or not, although there are some conditional results that give interesting information. It has been shown [4, 12] that if CLAUSE-INF is compilable, then NP $\subseteq$ P/poly holds, which implies [11] that $\Sigma_2^p = \Pi_2^p$, i.e., the Polynomial Hierarchy collapses, an event that – although not as strong as P = NP – is widely conjectured to be false. A similar result can be easily proven for the more general problem $T \models W$, where W is any propositional formula. If this problem is compilable, then P = NP, as tautologies could be detected in polynomial time compiling the empty T.

Nevertheless, to give a formal proof that a problem is not compilable is quite difficult. As an example, proving that CLAUSE-INF is not compilable implies P $\neq$ NP (a necessary and sufficient condition for compilability is shown in [4]).

In fact, this is a typical situation when dealing with non-compilability of problems, that we can parallel with the traditional notion of polynomial-time intractability: Proving intractability is usually done proving NP-hardness, which means that a problem does not have polynomial-time algorithms provided P $\neq$ NP.

Summing up, proving compilability can be done by exhibiting polynomials according to Definition 2.1, while non-compilability is proven only conditionally. Hence, in the following of the paper we will use the term "non-compilable problem", by meaning this conditional result (i.e., compilability would imply the collapse of the PH).

3. QUERIES COMPILABLE TO P

In this section we present the main result of the paper concerning compilability. We give a sufficient condition for compilability to P in terms of syntactic restrictions of SO formulae.

Lists of variables or predicates are denoted in boldface. In the following, $\delta(\mathbf{X})$ denotes that all free FO variables in formula δ are in the list $\mathbf{X}$. Moreover, $\delta(\mathbf{M})$ denotes that predicate symbols in the list $\mathbf{M}$ may occur in δ.

First of all, we present a result which will be useful in the following.

LEMMA 3.1. *Let ψ be a SO query of the form:*

$$\forall \mathbf{M}\Big\{\alpha(\mathbf{M}) \ \vee \ \forall \mathbf{X}\ \Big[\beta(\mathbf{X}) \ \vee \ \gamma(\mathbf{X},\mathbf{M})\Big]\Big\} \qquad (7)$$

where α is an arbitrary SO formula in which no free FO variables occur, and β and γ are arbitrary FO formulae, with β not containing predicates in the list $\mathbf{M}$.

Formula (7) *is equivalent to:*

$$\forall \mathbf{X}\ \beta(\mathbf{X}) \ \vee \ \mathit{view}(\mathbf{X}), \qquad (8)$$

where $\mathit{view}(\mathbf{X})$ is defined as:

$$\forall \mathbf{M}\ \alpha(\mathbf{M}) \ \vee \ \gamma(\mathbf{X},\mathbf{M}). \qquad (9)$$

PROOF. *Since variables in $\mathbf{X}$ do not occur in the sub-formula α, and since variables in α can be safely renamed to avoid conflicts with variables in $\mathbf{X}$, formula ψ can be easily rewritten as:*

$$\forall \mathbf{M}\Big\{\forall \mathbf{X}\ \Big[\beta(\mathbf{X}) \ \vee \ \alpha(\mathbf{M}) \ \vee \ \gamma(\mathbf{X},\mathbf{M})\Big]\Big\}.$$

Furthermore, $\forall \mathbf{X}$ can be switched with $\forall \mathbf{M}$, and since no predicate in $\mathbf{M}$ occurs in β, the latter quantifier can be pushed in, thus obtaining:

$$\forall \mathbf{X}\Big\{\beta(\mathbf{X}) \ \vee \ \forall \mathbf{M}\ \Big[\alpha(\mathbf{M}) \ \vee \ \gamma(\mathbf{X},\mathbf{M})\Big]\Big\}.$$

By using the definition of $\mathit{view}(\mathbf{X})$ as in (9)*, the thesis follows.* □

Lemma 3.1 captures several intuitions:

- Complex (possibly quantified) subformulae can be treated as single components, like $\alpha()$, $\beta()$, and $\gamma()$.
- This is very useful for modeling the off-line computation, which may involve all predicates, except for the on-line ones. As an example in formula (3) concerning CONJ-INF, the off-line computation involves $MOD(M, E_T)$ and $M(X)$, cf. (5).
- In query (7) it is possible to separate the SO and fixed parts from the varying part.

  This allows us to process the former in the off-line phase (by means of the view (9)), and to evaluate the rewritten query (8) in the on-line phase.

In this paper we make the assumption that the size of the query is fixed with respect to the size of the database, which is implied by the *data complexity* (cf. [19]) assumption. As a consequence, the arity of $\mathbf{X}$ is a constant, and polynomial space in the size of data is enough to represent $\mathit{view}(\mathbf{X})$. Furthermore, since (8) is an ordinary FO formula, its evaluation requires polynomial time in the size of data.

From these observations, the main theorem follows:

THEOREM 3.2. *Let ψ, α, β, γ, and view as in Lemma 3.1. If neither α nor γ contain varying predicates, then the problem specified by ψ is compilable to* P. *In particular, off-line reasoning is done by computing the view* (9)*, and on-line reasoning by evaluating the (rewritten) query* (8).

The above theorem is stated for SO queries starting with universal quantification, which are adequate for modeling deduction problems: In such problems, typically, a formula must be true in *all* models of a specified set. A dual theorem for SO queries starting with existential quantification (adequate, for example, for model checking problems) holds.

3.1 Examples

In this subsection we show some applications of the above theorem.

3.1.1 *From* coNP *to* P*: Deduction problems*

Theorem 3.2 immediately applies to the CONJ-INF problem (cf. Section 2), by assuming $\mathbf{M} = \{M\}$, $\mathbf{X} = \{X\}$, and:

- $\alpha(\mathbf{M}) = \neg MOD(M, E_T)$;
- $\beta(\mathbf{X}) = \neg J(X)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = M(X)$.

A generalization of CONJ-INF with the same fixed part is 2CNF-INF, in which the varying part is a formula in CNF with exactly two literals per clause (2CNF).

PROBLEM 3.1 (2CNF INFERENCE – 2CNF-INF).

Fixed part *Propositional formula T in CNF;*

Varying part *Propositional formula F in CNF with exactly two literals per clause (2CNF).*

Question *Does $T \models F$, i.e., is every model of T also a model of F?*

Representing a 2CNF formula F can be done simply by means of a binary (symmetric) relation R_F whose tuples correspond to the clauses in F. As an example, by referring to formula (1), the logically implied 2CNF formula $(\neg a \vee \neg c) \wedge (\neg b \vee c)$ can be encoded by the following relation:

R_F	
$\neg a$	$\neg c$
$\neg c$	$\neg a$
$\neg b$	c
c	$\neg b$

The problem $T \models F$ corresponds to the following $\mathrm{SO}_\forall$ formula:

$$\forall M\ \ MOD(M, E_T) \rightarrow \forall X_1\ \forall X_2\ (R_F(X_1, X_2) \rightarrow (M(X_1) \ \vee \ M(X_2))) \qquad (10)$$

which evaluates to true iff every model M of T contains at least one literal of every clause of F.

In this case the view is different: In particular, $\mathbf{M} = \{M\}$, $\mathbf{X} = \{X_1, X_2\}$, and Theorem 3.2 applies with:

- $\alpha(\mathbf{M}) = \neg MOD(M, E_T)$;

- $\beta(\mathbf{X}) = \neg R_F(X_1, X_2)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = M(X_1) \vee M(X_2)$.

As noted in [16], the generalization KCNF-INF of 2CNF-INF, in which the varying part is a CNF formula with exactly k literals per clause, is still compilable, as long as k is fixed with respect to the size of T. Query (10) can be modified to account for longer clauses simply by increasing the arity of R_F, and Theorem 3.2 still applies.

3.1.2 *From* coNP *to* P*: Graph coloring*

Theorem 3.2 can be applied to non-deductive problems. As an example, we consider the following *forced colors in a graph* problem:

PROBLEM 3.2 (FORCED COLORS IN A GRAPH).

Fixed part *Graph H;*

Varying part *Set of nodes S;*

Question *Is every node in S forced to be assigned a predetermined color (e.g., red) in each 3-coloring of H in which two fixed nodes, a and b, have a given color, e.g., red and green, respectively?*

Using unary predicates R, G, and B for the colors, and binary relation E_H for the graph, we can easily represent by means of a closed FO formula $3COL(R, G, B, E_H)$ the fact that R, G, and B are a 3-coloring of H. The "forced colors" query is as follows:

$$\forall R\ \forall G\ \forall B\ (3COL(R, G, B, E_H) \wedge R(a) \wedge G(b)) \rightarrow \forall X\ (S(X) \rightarrow R(X)). \tag{11}$$

Theorem 3.2 applies when $\mathbf{M} = \{R, G, B\}$, $\mathbf{X} = \{X\}$, and:

- $\alpha(\mathbf{M}) = \neg(3COL(R, G, B, E_H) \wedge R(a) \wedge G(b))$;
- $\beta(\mathbf{X}) = \neg S(X)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = R(X)$.

The intuitive meaning of the forced colors problem is to check whether choosing the color for a couple of nodes constrains a different set S of nodes (not known in advance) to have a specific color. Such a problem is an abstraction for several resource allocation problems, e.g., the *frequency assignment* problem in the context of telecommunications networks. In the frequency assignment problem nodes are transmitters, colors are frequencies, and an edge means that two nodes cannot be assigned the same frequency. In the query (11) we are basically asking whether assigning frequencies to a subset of the transmitters implies that other nodes (set S) must have a specific frequency. Clearly, query (11) can be generalized to any constant number of colors (frequencies).

As another example in the context of graphs we consider the *few blue neighbors* problem:

PROBLEM 3.3 (FEW BLUE NEIGHBORS).

Fixed part *Graph H;*

Varying part *Set T of nodes;*

Question *Does every node in T have at most one blue neighbor in each 3-coloring of H in which two fixed nodes, a and b, have a given color, e.g., red and green, respectively?*

A query for the few blue neighbors problem is the following:

$$\forall R\ \forall G\ \forall B\ (3COL(R, G, B, E_H) \wedge R(a) \wedge G(b)) \rightarrow \forall X\ \forall Y\ \forall Z\ (T(X) \wedge E_H(X, Y) \wedge E_H(X, Z) \wedge B(Y) \wedge B(Z)) \rightarrow Y = Z.$$

In this case, $\mathbf{M} = \{R, G, B\}$, $\mathbf{X}$ is $\{X, Y, Z\}$, and:

- $\alpha(\mathbf{M}) = \neg 3COL(R, G, B, E_H)$;
- $\beta(\mathbf{X}) = \neg T(X)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = \neg E_H(X, Y) \vee \neg E_H(X, Z) \vee \neg B(Y) \vee \neg B(Z) \vee (Y = Z)$.

Also the few blue neighbors problem abstracts resource allocation problems. As an example it can be viewed as a variant of the frequency assignment problem, in which we want transmitters in the set T to have at most one neighbor with a given frequency, because it may interfere with others.

3.1.3 *From* Π_2^p *to* P*: Circumscription*

In principle, we can apply Theorem 3.2 to any problem in the PH. As an example, we consider the CONJ-CIRC problem, defined as follows:

PROBLEM 3.4 (CONJUNCTION INFERENCE FROM CIRCUMSCRIPTION – CONJ-CIRC).

Fixed part *Propositional formula T in CNF;*

Varying part *Conjunction of literals $J = l_1 \wedge \cdots \wedge l_n$;*

Question *Does $CIRC(T) \models J$, i.e., is every model of the* circumscription *of T also a model of J?*

Circumscription [14] is a form of non-monotonic reasoning, and the models of $CIRC(T)$ are the *minimal* models of T, with respect to set containment (as an example, the minimal models of $a \vee b$ are $\{a\}$ and $\{b\}$, and the model $\{a, b\}$ is not minimal). As shown in [7], CONJ-CIRC (ignoring the distinction between varying and fixed parts) is Π_2^p-complete. We now show that it can be compiled to P. First of all we introduce the following $\mathrm{SO}_\forall$ formula (abbreviated as $MINMOD(M, E_T)$) which evaluates to true iff M represents a minimal model of T:

$$MOD(M, E_T) \wedge \forall N\ MOD(N, E_T) \rightarrow \neg[(\forall X\ A(X) \wedge N(X) \rightarrow M(X)) \wedge (\exists X\ A(X) \wedge M(X) \wedge \neg N(X))] \tag{12}$$

(here, A is a unary fixed relation storing the propositional variables occurring in T). Now, CONJ-CIRC corresponds to

the following query:

$$\forall M\ MINMOD(M, E_T) \rightarrow (\forall X\ J(X) \rightarrow M(X)),$$

and Theorem 3.2 applies with $\mathbf{M} = \{M\}$, $\mathbf{X} = \{X\}$, and:

- $\alpha(\mathbf{M}) = \neg MINMOD(M, E_T)$;
- $\beta(\mathbf{X}) = \neg J(X)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = M(X)$.

It is interesting to notice that off-line reasoning, i.e., to compute the view $\forall M\ MINMOD(M, E_T) \rightarrow M(X)$ is a Π_2^p-hard task. In this case, KC is even more convenient, since a Π_2^p-complete problem is transformed into a polynomial one.

4. QUERIES NON-COMPILABLE TO P

In this section we deal with non-compilability, in the conditional sense of Section 2. More precisely, we show the reasonableness of the assumptions of Theorem 3.2 concerning γ and α, by showing that the set of SO queries in which either assumption does not hold contains queries non-compilable to P.

THEOREM 4.1. *The set of formulae of Lemma 3.1 in which in γ there are varying predicates, contains queries non-compilable to* P.

PROOF. *By allowing formulae with varying predicates in γ, the problem* CLAUSE-INF *(cf. query* (4)*) can be modeled by taking $\mathbf{M} = \{M\}$, $\mathbf{X} = \{\}$, and:*

- $\alpha(\mathbf{M}) = \neg MOD(M, E_T)$;
- $\beta(\mathbf{X}) = \mathsf{false}$;
- $\gamma(\mathbf{X}, \mathbf{M}) = \exists Y\ C(Y) \wedge M(Y)$.

We remind that this problem, as mentioned in Section 2, is non-compilable to P. □

Before claiming the next result, we briefly describe a generalized version of circumscription (cf. Section 3.1.3), and the CONJ-(P;Z)-CIRC problem, useful for what follows.

PROBLEM 4.1 (CONJUNCTION INFERENCE FROM (P;Z)-CIRCUMSCRIPTION – CONJ-(P;Z)-CIRC).

Fixed part *Propositional formula T in CNF;*

Varying part *Conjunction of literals $\delta = l_1 \wedge \cdots \wedge l_n$;*

Question *Does $CIRC(T; P; Z) \models \delta$, i.e., is every $(P; Z)$-minimal model of T also a model of δ?*

In the $(P; Z)$-*circumscription* only a subset of the propositional variables (those in a specified set P) are minimized, while the remaining ones (in the complementary set Z) are left *floating* (cf. [14]). This leads to the notion of $(P; Z)$-*minimal* models, which coincide with minimal models described in Section 3.1.3 when $Z = \emptyset$. As an example, if $P = \{b\}$ and $Z = \{a\}$, then $a \vee b$ has only one $(P; Z)$-minimal model, i.e., $\{a\}$ (the minimal model $\{b\}$ is not $(P; Z)$-minimal).

The $\mathrm{SO}_\forall$ formula (abbreviated as $PZMINMOD(M, P, E_T)$) which evaluates to true iff M represents a $(P; Z)$-minimal model of T can be easily obtained by modifying formula (12) as follows:

$$\begin{aligned} MOD(M, E_T) \wedge \forall N\ MOD(N, E_T) \rightarrow \\ \neg[(\forall X\ P(X) \wedge N(X) \rightarrow M(X)) \wedge \\ (\exists X\ P(X) \wedge M(X) \wedge \neg N(X))], \end{aligned}$$

where, P is a unary relation storing all propositional variables to be minimized.

Hence, the CONJ-(P;Z)-CIRC problem can be immediately specified with the query:

$$\begin{aligned} \forall M\ PZMINMOD(M, P, E_T) \rightarrow \\ (\forall X\ J(X) \rightarrow M(X)). \end{aligned} \tag{13}$$

We are now ready to present the next theorem.

THEOREM 4.2. *The set of formulae of Lemma 3.1 in which in α there are varying predicates, contains queries non-compilable to* P.

PROOF. *Consider the* CONJ-(P;Z)-CIRC *problem with the P set varying; by allowing formulae of kind* (7) *with varying predicates in α, the query* (13) *matches conditions of Lemma 3.1 by taking $\mathbf{M} = \{M\}$, $\mathbf{X} = \{X\}$, and:*

- $\alpha(\mathbf{M}) = \neg PZMINMOD(M, P, E_T)$;
- $\beta(\mathbf{X}) = \neg J(X)$;
- $\gamma(\mathbf{X}, \mathbf{M}) = M(X)$.

However, it has been proven in [4] that in such a case $CIRC(T; P; Z) \models \delta$ is non-compilable to P*, even when T is a positive 2CNF formula and δ is a single literal.* □

We remark that the above theorems just deal with syntactic criteria. Their goal is not to provide a sharp separation between compilable and non-compilable problems, but rather to justify the assumptions of Theorem 3.2.

The similarity between CONJ-INF (3) and CLAUSE-INF (4) deserves further comments. In general, a SO formula can be put in prenex form, and its matrix in CNF, treating arbitrary SO sub-formulae in which no free FO variables occur (like α in Lemma 3.1) as single literals. A formula in this form is said to be in *prenex generalized CNF* (PGCNF). In all compilability examples of Section 3, queries are actually in PGCNF, and their matrices contain exactly one clause. On the other hand, we note that CLAUSE-INF (4) is also in PGCNF, but has two clauses.

It is actually possible to show PGCNF queries with two clauses, which are compilable to P. As an example, the

next query combines the two compilable problems of Section 3.1.2, having the same fixed part:

$$\forall R\ \forall G\ \forall B\ (3COL(R,G,B,E_H) \wedge R(a) \wedge G(b)) \rightarrow$$
$$[\forall X\ \forall Y\ \forall Z\ (T(X) \wedge E_H(X,Y) \wedge E_H(X,Z)$$
$$\wedge B(Y) \wedge B(Z) \rightarrow Y = Z)\ \wedge \forall W\ (S(W) \rightarrow R(W))].$$

The above query is of the form:

$$\forall \mathbf{M}\,\{\alpha\,(\mathbf{M})\ \vee\ [\,(\forall \mathbf{X_1}\ \beta\,(\mathbf{X_1})\ \vee\ \gamma\,(\mathbf{X_1},\mathbf{M}))\ \wedge\ (\forall \mathbf{X_2}\ \beta\,(\mathbf{X_2})\ \vee\ \gamma\,(\mathbf{X_2},\mathbf{M}))]\}\,,$$

and all problems of this kind are indeed compilable to P, as shown by the next theorem, which generalizes Theorem 3.2.

THEOREM 4.3 ($\wedge$-COMPOSITION). *Let ψ be a SO query of the form:*

$$\forall \mathbf{M} \bigwedge_i [\alpha_i\,(\mathbf{M})\ \vee\ \forall \mathbf{X_i}\ (\beta_i\,(\mathbf{X_i})\ \vee\ \gamma_i\,(\mathbf{X_i},\mathbf{M}))]\,, \qquad (14)$$

where α_i are arbitrary SO formulae in which no free FO variables occur, β_i and γ_i are arbitrary FO formulae, with β_i not containing predicates in $\mathbf{M}$, and α_i, γ_i do not contain varying predicates.

The problem specified by ψ is compilable to P. *In particular, off-line reasoning is done by computing the views $view_i(\mathbf{X_i})$, defined as:*

$$\forall \mathbf{M}\ \alpha_i\,(\mathbf{M})\ \vee\ \gamma_i\,(\mathbf{X_i},\mathbf{M})\,,$$

and on-line reasoning can be done by evaluating the (rewritten) query:

$$\bigwedge_i \{\forall \mathbf{X_i}\ \beta_i\,(\mathbf{X_i})\ \vee\ view_i(\mathbf{X_i})\}\,.$$

PROOF. *Because of the presence of the universal quantifier and the outermost connective $\wedge$, by pushing $\forall\mathbf{M}$ in, formula (14) can be safely rewritten as $\bigwedge_i \psi_i$ where each ψ_i is of the form (7). Now, query ψ can be rewritten using the same arguments of the proof of Lemma 3.1.* □

The above theorem says that, when we have two or more problems which are "independently" compilable to P, then their combination is compilable to P as well. Obviously, the theorem does not apply to the CLAUSE-INF problem, because in (4) the scope of the FO variable X spans both clauses of the matrix. In particular, it seems that the presence of the same FO variable in both a fixed and a varying predicate in different clauses is a plausible predictor for non-compilability of a query.

5. QUERIES COMPILABLE TO OTHER CLASSES

In general, KC deals with making on-line reasoning more efficient, but not necessarily polynomial-time. Hence, a definition more general than Definition 2.1 is that of *problem compilable to* C, where C is an arbitrary complexity class:

DEFINITION 5.1 (PROBLEM COMPILABLE TO C). *A problem π, which is split into a fixed and a varying part, FIX and VAR respectively, is said to be* compilable to C *if there exists a polynomial p and an algorithm ASK such that for each instance KB of FIX there is a data structure D_{KB} such that:*

1. *$|D_{KB}| \leq p(|KB|)$;*
2. *for each instance f of VAR the call $ASK(D_{KB}, f)$ returns yes iff $\langle KB, f\rangle$ is a "yes" instance of π;*
3. *$ASK(D_{KB}, f)$ is in the complexity class* C.

As an example, it makes sense to compile a problem which is Π_2^p-complete to a coNP-complete one, if it is non-compilable to P. To this end, Theorem 3.2 is useless, because it only deals with compilation to P. On the other hand, if we can modularize a SO query as a formula of the form:

$$\mathbf{Q_S M}\ \mathbf{Q_F X}\ \eta(\mathbf{M},\mathbf{X},\delta(\mathbf{X})) \qquad (15)$$

where $\mathbf{Q_S}$ and $\mathbf{Q_F}$ are, respectively, lists of SO and FO quantifiers, δ is an arbitrary SO formula in which neither on-line predicates nor predicates in $\mathbf{M}$ occur, and η is SO, then the evaluation of $\delta(\mathbf{X})$ can be done off-line.

Several interesting deduction problems can indeed be specified in such a form. As an example, another variant of the CLAUSE-INF problem is the CLAUSE-GCWA problem, defined as follows:

PROBLEM 5.1 (CLAUSE INFERENCE IN THE GENERALIZED CLOSED-WORLD ASSUMPTION – CLAUSE-GCWA).

Fixed part *Propositional formula T in CNF;*

Varying part *Disjunction of literals $C = l_1 \vee \ldots \vee l_n$;*

Question *Does $GCWA(T) \models C$, i.e., is every model of $GCWA(T)$ also a model of C?*

The *generalized closed-world assumption* (GCWA) of a propositional formula T is a form of non-monotonic reasoning defined in [15] as follows:

$$GCWA(T) = T \cup \{\neg x \mid \text{variable } x \text{ is } \mathsf{false} \text{ in all minimal models of } T\}$$

As an example, the GCWA of formula $T = a \vee b$ is T itself, since both a and b are true in at least one minimal model of T.

As shown in [7], CLAUSE-GCWA is Π_2^p-complete; moreover, if T is fixed, compilation to P is not possible, cf. [4]. There is some room for compilation to an intermediate class, such as coNP, using the following idea. All negative literals which are true in all minimal models of T can be computed during the off-line phase, and can obviously be represented in a polynomial-size structure. On-line reasoning will use such literals as well as the original formula T, and can be done by means of a coNP computation.

The following view $GCWA\_LIT(Y)$ represents off-line reasoning, because it stores all negative literals Y which are

true in all minimal models of T:

$$\forall X \ \ (A(X) \wedge E_T(X,Y) \wedge E_T(Y,X)) \rightarrow \\ (\forall N \ MINMOD(N, E_T) \rightarrow N(Y))$$

(here, A is a unary fixed relation storing the propositional variables occurring in T). Note that Y is a negative literal, because it is the negation of X, which is a variable of T.

Now, CLAUSE-GCWA can be written as:

$$\forall M \ [MOD(M, E_T) \ \wedge \ (\forall Y \ GCWA\_LIT(Y) \rightarrow M(Y))] \rightarrow \\ (\exists X \ C(X) \wedge M(X)),$$

which is indeed in the form (15), with $\delta(\mathbf{X}) = GCWA\_LIT(Y)$.

Another interesting problem specifiable by means of a formula of the form (15) is the WIDTIO approach to knowledge base revision [20]. In this case, the problem is to check whether a clause q follows from a formula T revised with another formula p that makes T inconsistent. The view is the set of clauses in T which "survive" after the revision. The Π_2^p-complete query can be compiled to a coNP-complete one.

It is worth noting that formulae of the kind (15) subsume those of Theorem 3.2. They are also less precise, because they capture formulae like (4): just take $\delta(\mathbf{X})$ as the empty formula.

6. CONCLUSIONS

In this paper we have studied Knowledge Compilation in the context of relational databases. Both fixed and varying parts of the instance are represented as relations, and the inference operator is represented as a query in second-order logic. Our main result concerning compilability is the definition of a syntactic restriction of second-order formulae which guarantees the existence of a polynomial-size view and a first-order equivalent rewritten query, thus allowing on-line reasoning to be done in polynomial time.

We have applied such a result to a variety of deduction problems, with underlying complexity at various levels of the Polynomial Hierarchy. Non-deductive problems concerning graphs have also been addressed.

The syntactic restriction has been motivated by showing that relaxing it we obtain a set of second-order queries containing queries non-compilable to P.

Finally, we have addressed the issue of compilability to higher classes in the Polynomial Hierarchy, by showing examples of Π_2^p-complete deduction problems which are compilable to coNP-complete ones.

Our approach is somewhat limited because our target language (for problems compilable to P) is first-order logic. This implies that some compilable problems, like, e.g., "cycles in a Hamiltonian reduction", defined in [2], cannot be captured, because they involve on-line reasoning which is not first-order expressible. To this end, we plan to investigate extensions of the target language to classes of formulae in fixpoint logic.

We also aim to characterize in a more detailed way the "frontier" between compilable and non-compilable problems, e.g., by adding finer-grained constraints to the conditions of Theorems 4.1 and 4.2.

Acknowledgements

The authors are grateful to the anonymous reviewers, for useful comments which helped improving the paper.

7. REFERENCES

[1] R. Ben-Eliyahu and R. Dechter. Default reasoning using classical logic. *Artificial Intelligence*, 84(1–2):113–150, 1996.

[2] M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications—The European Journal on Artificial Intelligence*, 10:137–150, 1997.

[3] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. Technical Report DIS 24-97, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", November 1997. To appear in *Information and Computation*.

[4] M. Cadoli, F. M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? *Artificial Intelligence*, 88(1–2):215–251, 1996.

[5] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[6] A. Darwiche and P. Marquis. A perspective on knowledge compilation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 470–475, 2001.

[7] T. Eiter and G. Gottlob. Propositional circumscription and extended closed world reasoning are Π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.

[8] R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.

[9] G. Gogic, H. A. Kautz, C. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 862–869, 1995.

[10] G. Gottlob, P. Kolatis, and T. Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. In *Proceedings of the Fortyfirst Annual Symposium on the Foundations of Computer Science (FOCS 2000)*. IEEE Computer Society Press, 2000.

[11] R. M. Karp and R. J. Lipton. Some connections between non-uniform and uniform complexity classes. In *Proceedings of the Twelfth ACM Symposium on Theory of Computing (STOC'80)*, pages 302–309, 1980.

[12] H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 786–793, 1992.

[13] P. G. Kolaitis and M. N. Thakur. Logical definability of NP optimization problems. *Information and Computation*, 115(2):321–353, 1994.

[14] V. Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 121–127, 1985.

[15] J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of the Sixth International Conference on Automated Deduction (CADE'82)*, pages 292–308, 1982.

[16] Y. Moses and M. Tennenholtz. Off-line reasoning for on-line efficiency: knowledge bases. *Artificial Intelligence*, 83:229–239, 1996.

[17] A. Nerode, R. T. Ng, and V. S. Subrahmanian. Computing circumscriptive databases. I: Theory and algorithms. *Information and Computation*, 116:58–80, 1995.

[18] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43:193–224, 1996.

[19] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the Fourteenth ACM SIGACT Symposium on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[20] M. Winslett. *Updating Logical Databases.* Cambridge University Press, 1990.