A Survey on Knowledge Compilation

Marco Cadoli and Francesco M. Donini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: {cadoli|donini}@dis.uniroma1.it

Knowledge compilation is an AI technique for addressing computationally demanding reasoning problems. In this paper we survey recent results in knowledge compilation of propositional knowledge bases. We first define and limit the scope of such a technique, then we survey exact and approximate knowledge compilation methods. We include a discussion of compilation for non-monotonic knowledge bases.

Keywords: Knowledge Representation, Efficiency of Reasoning

1. Introduction

Knowledge Representation (KR) is based on the idea that an agent stores what it knows in a *knowledge base*, typically using a logical formalism. The knowledge base is then used, through specific algorithms for querying, to extract information which is implicitly stored in it.

It is well known that deduction in a logical formalism is very much demanding from a computational point of view. In fact, problems in logic are known among computer scientists for their high computational complexity. As an example, if the knowledge base and the query are represented in first-order logic, then checking whether the query is logically entailed by the knowledge base or not is the prototypical recursively enumerable, non-recursive problem. If the reasoner represents its knowledge in formulae of propositional logic, still using a sound and complete procedure for logical entailment, then it faces a coNP-complete problem, i.e., a problem which probably requires exponential time to be solved.

Knowledge compilation is one of the techniques that have been proposed for addressing such computational difficulties. The central observation is that in KR reasoning problems, the input to a problem is conceptually split in two parts: a knowledge base and a query. Typically, the knowledge base is not modified very often, and the same knowledge base is used to answer many queries. The idea of knowledge compilation is to split query answering into two phases:

- in the first one the knowledge base is preprocessed, thus obtaining an appropriate data structure (such a phase is sometimes called *off-line reasoning*);
- in the second phase, the query is actually answered using the output of the first phase (such a phase is sometimes called *on-line reasoning*).

The goal of preprocessing is to make on-line reasoning computationally easier with respect to reasoning in the case when no preprocessing at all is done. Intuitively, the effort spent in the preprocessing phase pays off when its computational cost is amortized over the (facilitated) answer to many queries.

Knowledge compilation is not the only proposal made to cope with intractable query answering in knowledge bases. Two orthogonal techniques are *language restriction* and *theory approximation*. In language restriction, knowledge and/or queries are formed with a subset of the entire language (usually a syntactic restriction). On this subset, tractability is guaranteed. Also in knowledge compilation some restrictions on the language are usually assumed, e.g., clausal form for queries. Language restriction and knowledge compilation can be also combined, e.g., in the compilation of Horn propositional knowledge bases. In theory approximation, only a part of the entailed and non-entailed queries are correctly answered, while the others are given a "don't know" answer. The correctly answered queries usually correspond to a tractable entailment problem. Also theory approximation can be combined with knowledge compilation, e.g., compiling a propositional knowledge base KB into a Horn formula whose models are a subset or a superset of the models of KB.

Another issue related to knowledge compilation, which was discussed in the AI literature (cf., e.g., [36, 4]), is *vivification* of knowledge. Vivification is the process of making reasoning tractable by using both preprocessing and default conclusions, at the expense of losing completeness, and, sometimes, soundness.

AI Communications 0 () 0
ISSN 0921-7126 / $8.00 © , IOS Press

1.1. Overview of the paper

The goal of this paper is to survey recent advances in knowledge compilation, considering both exact and approximate knowledge compilation. We put more emphasis on the theoretical aspects, and we just briefly comment on the experimental ones. We analyze mainly compilation of classical, propositional knowledge bases, although we also consider non-monotonic and mention non-propositional ones, respectively.

Theoretical studies show that the result of the compilation can have an unacceptably large size, i.e., its size is exponential in the size of the original knowledge base. In Section 2 we introduce main definitions, then summarize and comment some theoretical results on the limits of knowledge compilation.

Although not every knowledge base is compilable, it still makes sense to devise compilation methods that may give reasonable experimental results for some classes of knowledge bases. In Section 3 we report on such exact methods for compilation of propositional knowledge bases, and their experimental evaluation.

The possible exponential growth of the size of the compiled knowledge base lead several researchers to explore the possibility of compiling the knowledge base into some data structure that *approximates* the initial knowledge base, as an example, giving up completeness of reasoning. In Section 4 we survey approximate compilation methods for such knowledge bases.

Sometimes the purpose of knowledge compilation is not restricted to polynomial-time on-line reasoning. A typical case is compilation of *non-monotonic* knowledge bases into classical, monotonic ones. This can still be reasonable, since reasoning in non-monotonic knowledge bases is usually computationally much more demanding than reasoning in monotonic ones. We analyze compilation of non-monotonic knowledge bases into classical ones in Section 5.

Other forms of knowledge compilation are also possible. Firstly, knowledge compilation can be done also in the case of reasoning tasks which are already polynomial-time solvable, such as reasoning in the context of Horn propositional formulae. In such a case, the goal is to optimize the knowledge base, as an example, by minimizing its size. Secondly, most techniques for knowledge compilation of a knowledge base yield logical formulae, e.g., prime implicants, as a result of the off-line reasoning part. However, some of them yield data structures which are not formulae, but sets of models structured in some complex way. In Section 6 we consider compilation that does not yield knowledge bases, and compilation of knowledge bases which already support polynomial-time inference algorithms. In the same section we briefly address the related field of fixed parameter tractability.

Finally a brief conclusion is drawn in Section 7.

2. Knowledge compilation: Terminology

In the paper we assume the reader is familiar with the basic concepts of Propositional Logic and Automated Theorem Proving. For an introduction to the subject, see for example [24]. Throughout the paper we interchangeably consider a knowledge base both as a set of formulae and as one formula made of the conjunction of all formulae in such a set. We denote the *size* of a knowledge base (or a formula) KB, i.e., the number of distinct occurrences of propositional variables, as $|KB|$.

In the paper we also assume that the reader has some basic notions of computational complexity concepts; we refer to [25, 46] for a thorough introduction in the field of complexity. In particular, we refer to complexity classes P, NP, coNP, Σ_2^p, Π_2^p, Σ_3^p, PH. The known relations among them are the following (i = 2,3):

$$\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}; \qquad \mathrm{NP}, \mathrm{coNP} \subseteq \Sigma_2^p \cap \Pi_2^p;$$
$$\Sigma_i^p, \Pi_i^p \subseteq \mathrm{PH}.$$

All such inclusions have been conjectured to be strict.

Preprocessing is quite common in Computer Science. As an example, compilers usually optimize object code by performing simple operations, simplifying tests, exchanging loops, etc. Another example can be the preprocessing of a graph to obtain a data structure that allows for fast node reachability test. The idea is that the time spent during the preprocessing will allow the compiled code to gain (much more) time during the processing of data.

However, in Computer Science preprocessing is usually done for problems which are already solvable in polynomial time, e.g., node reachability test in a graph. What characterizes the study of such techniques in the context of AI and KR is that reasoning problems are often NP-hard, i.e., either NP-complete or coNP-complete, or even more complex. This has some significant computational consequences, especially on the size of the output of the off-line reasoning part.

We now introduce a very simple reasoning problem which we use as running example. The LITERAL EN-

TAILMENT problem is specified by means of 1) its input, and 2) the question we are expected to solve, in the following way:

INSTANCE: Finite set L of Boolean variables, a propositional formula in Conjunctive Normal Form T, and a literal l (both T and l are built using variables in L.)
QUESTION: Is it true that all models of T are models of l (i.e., that $T \models l$)?

In the CLAUSE ENTAILMENT problem, the instance specifies a clause γ (disjunction of literals), as opposed to a single literal.

Both LITERAL ENTAILMENT and CLAUSE ENTAILMENT are coNP-complete problems, although they are quite different from the point of view of preprocessing, as we are going to see in this section.

The usual way to present a problem as a pair INSTANCE/QUESTION does not describe which part of INSTANCE is fixed. Therefore, we have to split the instance into a fixed and a varying part. For example, if T represents a knowledge base, we may assume that it remains stable over a long period of time, hence it makes sense to preprocess it; on the other hand, the query which is posed to the knowledge base, represented either by literal l or by clause γ, will be different for each instance. Here is the terminology we propose for addressing such preprocessing aspects:

fixed part of a problem: The part that goes through preprocessing in a problem that is compiled.
Example: A propositional formula T in CNF.

varying part of a problem: The part that does not go through preprocessing.
Example: A literal l.
Example: A clause γ.

(**structured**) **problem:** A triple consisting of the question that we ultimately want to solve, its fixed part, and its varying part.
Example: $[T \models l, T, l]$.
Example: $[T \models \gamma, T, \gamma]$.

The goal of knowledge compilation is to make online reasoning easier with respect to the "all-at-once" setting. For example, as LITERAL ENTAILMENT is coNP-complete, a reasonable expectation is to be able to solve $[T \models l, T, l]$ on-line in time polynomial in $|T| + |l|$. As a matter of fact, this is possible, because we can record on a table, for each literal l occurring in T, whether $T \models l$ or not. The size of the table is in $O(n)$, where n is the cardinality of the alphabet L of T ($n \in O(|T|)$); the table can be consulted in $O(n)$ time (sublinear time could be achieved with custom data structures). Building the table amounts to solve $O(n)$ instances of a coNP-complete problem; but, as we already mentioned, we are not concerned with the off-line computational burden.

In the above "compilation method" for $[T \models l, T, l]$ two desiderata are met:

1. the preprocessing phase outputs a data structure whose size grows polynomially with respect to the size of the above mentioned fixed part (the knowledge base);
2. on-line reasoning takes time polynomial in the size of the data structure plus the size of the varying part (the query).

In Section 2.1 we show that, at least for some important problems, both requirements are unlikely to be achieved at the same time. However, such desiderata give us guidelines for defining the scope of knowledge compilation. We also believe that the same preprocessing should facilitate the answer to a whole class of queries – not just one. Finally, even if the compilation process can take a substantial amount of time, it must always terminate. These considerations lead to the following informal definition of knowledge compilation:

> *Knowledge compilation* deals with methods of processing off-line a knowledge base in such a way that the output of such a preprocessing can be used to speed-up on-line answering for a class of queries. The preprocessing should take a finite amount of time.

Note that this definition excludes techniques such as lemmaizing in theorem proving [1] as knowledge compilation techniques, unless the same lemmata are reused in many different proofs, and the theorem prover stops in a finite amount of time.

We limit the scope of most of this survey to *propositional* knowledge bases, and to compilations that change the complexity class of query answering (e.g., from coNP-complete to P, from Π_2^p-complete to coNP-complete), and whose output is still a logical formula. In Section 6 we consider forms of compilation outside such bounds.

Before surveying actual methods for knowledge compilation, we discuss in the next subsection some theoretical limits to knowledge compilation that apply to any possible compilation.

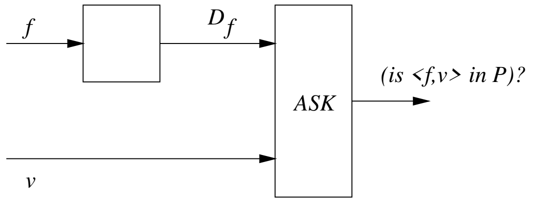

Fig. 1. Compilation of a problem

2.1. Theoretical limits to knowledge compilation

Recalling the distinction of fixed and varying parts of a problem, a possible definition for "compilable" problems is the following.

Definition 1 (cf. [8]) *A problem* $[P,\ F,\ V]$ *is* compilable *if there exist two polynomials* p_1, p_2 *and an algorithm* ASK *such that for each instance* f *of* F *there is a data structure* D_f *such that:*

1. $|D_f| \leq p_1(|f|)$*;*
2. *for each instance* v *of* V *the call* $ASK(D_f, v)$ *returns* `yes` *iff* $\langle f, v\rangle$ *is a "yes" instance of* P*;*
3. $ASK(D_f, v)$ *requires time* $\leq p_2(|v| + |D_f|)$.

Figure 1 gives a pictorial description of Definition 1. In the above definition:

1. the data structure D_f represents the output of off-line reasoning, and it is required to occupy space polynomial in the size of the fixed part;
2. the algorithm ASK represents the on-line reasoning part, and it is required to be sound and complete;
3. ASK is also required to run in polynomial time in its inputs.

According to Definition 1, $[T \models l, T, l]$ is compilable, since the space of its possible queries has size polynomial in $|T|$. This is just a sufficient condition, as a problem might be compilable even if the condition is not true. Let's consider the problem $[T \models \delta, T, \delta]$, where δ is a conjunction of literals, and both T and δ are built on a finite set L of literals (the "all-at-once" version is coNP-complete). The possible conjunctions are at least 2^n (n being the size of L), but the table mentioned in the context of $[T \models l, T, l]$ still suffices for a polynomial-time on-line answer, as for each set $\{l_1, \ldots, l_m\}$ of literals, $T \models l_1 \wedge \cdots \wedge l_m$ iff $T \models l_1$, $\ldots, T \models l_m$.

For a problem such as $[T \models \gamma, T, \gamma]$ (γ being a clause) it is less obvious how to define a compilation method (note that the possible clauses are at least 2^n). To see this, let's consider the following preprocessing: Compute the conjunction $PI(T)$ of all prime implicates of T. As $PI(T)$ is logically equivalent to T, we have that $PI(T) \models \gamma$ iff $T \models \gamma$; moreover, the former check can be done in time $O(|PI(T)| + |\gamma|)$ (i.e., we have proven the existence of a polynomial p_2 according to Definition 1). Unfortunately, it is well known that the number of prime implicates of a formula T can be exponential in the size of the alphabet of T (cf. Section 3.1), i.e., we still don't have a polynomial p_1. As a consequence such a preprocessing does not prove that $[T \models \gamma, T, \gamma]$ is compilable.

Referring to the last example, it is interesting to comment on the importance of the first item in Definition 1, i.e., the fact that the compiled knowledge base must remain of size polynomial in the size of the original knowledge base[1]. Suppose we dropped such a restriction: we should obviously replace the third property with the assertion "the ASK algorithm requires time polynomial in $|v| + |f|$, i.e., the size of the original knowledge base plus the size of the query". With such a definition, it would be possible to store in a data structure the result of test $T \models \gamma$ for every clause γ. Such a data structure would be of size *exponential* in the number of variables, but every clausal query could be answered in time polynomial in the size of the original knowledge base using (logarithmic-time) binary search. In other words, the problem $[T \models \gamma, T, \gamma]$, γ being a clause, would be trivially compilable.

The definition of non-compilable problem follows from Definition 1: $[P,\ F,\ V]$ is non-compilable if there are no polynomials p_1, p_2 with the above properties. A necessary (but not sufficient) condition for non-compilability is that the set of possible varying parts of the input V is super-polynomial in the size $|F|$ of the fixed part. Intuitively, if a polynomial p_1 with the properties mentioned in Definition 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.

As for the $[T \models \gamma, T, \gamma]$ problem, we still do not know whether it is compilable or not, although there are some conditional results that give interesting information. It has recently been shown [8, 34] that if

[1] We thank Pierre Marquis for this remark.

$[T \models \gamma, T, \gamma]$ is compilable, then $\Sigma_2^p = \Pi_2^p$, 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 $[T \models W, T, W]$, 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 knowledge base.

Nevertheless, to give a formal proof that a problem is not compilable is quite difficult. As an example, proving that $[T \models \gamma, T, \gamma]$ is not compilable implies P $\neq$ NP (a necessary and sufficient condition for compilability is shown in [8]).

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 1, while non-compilability is proven only conditionally.

3. Exact compilation

Exact compilation methods try to find a representation of all the theorems of the initial (propositional) knowledge base. We can classify proposals for exact knowledge compilation in three main methods:

1. use prime implicants or prime implicates (Section 3.1);
2. add to the knowledge base only those prime implicates that make any deduction possible by unit resolution (Section 3.2);
3. use prime implicates with respect to a tractable theory (Section 3.3).

3.1. Prime implicants and prime implicates

The simplest proposals on exact knowledge compilation stem from the fact that every knowledge base has normal forms (namely, conjunctive and disjunctive normal forms, CNF and DNF, respectively) from which consequences can be easily computed. We first consider the DNF.

An *implicant* of a knowledge base KB is a conjunction of literals D such that $D \models KB$ and D does not contain two complementary literals. Models of a knowledge base are one-to-one with implicants in which each variable appears exactly once, as a positive literal if it is assigned true, as a negative literal otherwise. It is often convenient to see a conjunction of literals as a set of literals. We make use of such an alternative view in the following definition of prime implicants.

Definition 2 *A* prime implicant *of a knowledge base KB is an implicant D of KB such that for every other implicant D' of KB, $D' \not\subset D$.*

Making a disjunction of all prime implicants $D_1, \dots, D_k$ of a knowledge base KB, one obtains a DNF formula $D_1 \vee \cdots \vee D_k$ which is equivalent to KB, and such that for every query Q, $KB \models Q$ iff for every prime implicant D_i, $D_i \models Q$. If Q is in CNF (a conjunction of clauses), this amounts to verify that every clause of Q has a non-empty intersection with every prime implicant of KB. Hence entailment of CNF queries can be computed in time polynomial in the size of the set of prime implicants plus the size of the query.

Dual to methods based on the DNF, there are knowledge compilation methods based on the CNF of formulae. An *implicate* of a knowledge base KB is a disjunction of literals C (a clause) such that $KB \models C$ and C does not contain two complementary literals.

Clauses can be seen as sets of literals. If also models are seen as sets of literals, then for each model M of KB an implicate C of KB must contain at least one literal of M. In other words, an implicate is a hitting set of the models of a knowledge base. Minimal implicates, called *prime implicates*, are minimal (with respect to set containment) hitting sets. Most of the compiling methods targeted for speeding up query answering rely on the prime implicates of a knowledge base.

Definition 3 *A* prime implicate *of a knowledge base KB is an implicate C of KB such that for every other implicate C' of KB, $C' \not\subset C$.*

Making a conjunction of all prime implicates $C_1, \dots, C_k$ of a knowledge base KB, one obtains a CNF formula $C_1 \wedge \cdots \wedge C_k$ which is equivalent to KB. For every CNF query Q, $KB \models Q$ iff for each non-tautological clause C' of Q there is a prime implicate C of KB such that $C \models C'$, i.e., $C \subseteq C'$ if clauses are seen as sets of literals. Hence entailment of CNF queries can be computed in time polynomial in the size of the set of prime implicates plus the size of the query.

Intuitively, all prime implicates of a CNF knowledge base can be generated by repeatedly resolving clauses of the knowledge base (each resolvent is an implicate) and discarding implicates which are not prime. However, this method may require too many resolutions to be performed. Research on algorithms for computing prime implicates started a long time ago: some algorithms can be found in [55, 54, 48, 33, 38, 14, 12]. Most of them rely on the assumption that the knowledge base is in CNF. Ngair [45] proposes a more general algorithm for computing prime implicates that requires the knowledge base to be a conjunction of DNF formulae.

From a theoretical study, the numbers of prime implicants and prime implicates of a knowledge base with n variables were shown [13] to be exponential in n in the worst case. Schrag and Crawford [50] made an experimental study of the number of implicates and prime implicates for CNF knowledge bases of increasing size, at different values of the ratio (number of clauses)/(number of variables). To explain the importance of this ratio, we briefly recall some recent results which are outside knowledge compilation. The clauses/variables ratio has been studied in many experiments [32] about the difficulty of solving satisfiability problems. It has been found that the knowledge bases in 3CNF can be roughly classified in three types according to their clauses/variables ratio:

1. Below a critical value, the few clauses in the knowledge base can be easily satisfied. These knowledge bases are called "underconstrained".
2. Above the critical value, a refutation involving part of the so many clauses can be easily found. These knowledge bases are called "overconstrained".
3. Around the critical value of 4.3, it is hard to find both a satisfying truth assignment, and a refutation. These knowledge bases are called "critically-constrained", and are those for which the probability of being satisfiable is around 0.5 [53].

Coming back to knowledge compilation, Schrag and Crawford found that as the size of the knowledge base increases, the number of prime implicates for each given length shows a peak in correspondence of critically-constrained knowledge bases. Therefore experiments suggest to use such knowledge bases as benchmarks to test knowledge compilation methods based on prime implicates.

Most methods for generating prime implicates can be adapted for the computation of prime implicants. Instead, Schrag [49] proposes a method for computing prime implicants based on a variant of the Davis-Putnam (DP) procedure. The DP procedure is used to test satisfiability of a CNF formula; if the formula is satisfiable, it outputs a truth assignment – or equivalently, a set of literals – satisfying the formula. Such a set of literals can be seen as an implicant of the formula. Since the DP procedure searches through all possible assignments, it eventually produces all implicants, from which prime implicants can be selected. Schrag tests his method for critically-constrained 3CNF knowledge bases.

Dechter and Rish [15] propose to add implicates to a CNF knowledge base by performing resolutions based on a given ordering of variables. After the addition, one can generate models of the knowledge base in time linear in the compiled knowledge base. Hence, their compilation method is more focused on facilitating model generation than query answering.

3.2. Unit-resolution-complete compilations

Since the number of prime implicates is exponential in many cases, del Val [16] proposes an enhanced method using prime implicates. Let us consider some preliminary aspects that show intuitively the advantages of such a method.

In the previous subsection we saw that entailment between a CNF query and a prime-implicates-compiled knowledge base can be done by checking whether each query clause is contained in a prime implicate. Now, this check for containment is a form of *unit-resolution* refutation, whose definition we recall now. Unit resolution is a form of resolution where one of the two clauses to be resolved is a single literal. It is sound, but incomplete, i.e., not all refutations can be found by unit resolution. Now, negating a clause in the query yields a set of negated literals, and by unit resolution one obtains the empty clause if and only if there is a prime implicate made by a subset of the literals in the clause.

However, unit resolution can do more than simple set containment. By substituting the set-containment check with unit resolution refutation, one does not need to keep all prime implicates, but only a subset of them from which *by unit resolution* every other implicate can be obtained. Since every unit-resolution refutation needs polynomial time in the size of the initial clauses to be refuted, this method turns also a coNP-complete problem into a problem solvable in polynomial time with respect to the size of the formula produced by preprocessing.

del Val proposes to use any prime-implicates-generating algorithm, filtering out all prime implicates that can be reconstructed by unit resolution during the on-line phase. Recall that most algorithms for finding implicates of a CNF knowledge base work by performing repeated resolutions. Prime implicates kept by del Val's method are only *merge resolvents*. A resolvent of two clauses C_1 and C_2 is a merge resolvent if after performing resolution on a literal l, the resolvent contains two identical literals, i.e., $C_1 - \{l\}$ and $C_2 - \{\neg l\}$ (seen as sets) have a non-empty intersection.

Mathieu and Delahaye [41] define in about the same way *redundant* clauses, which are clauses which could be derived from the others by performing a form of unit resolution which they call "forward chaining" – basically, the chaining made in knowledge bases constituted by rules (definite Horn clauses).

To see why merge resolvents cannot be discarded, we give an example, following Mathieu and Delahaye's approach. Suppose there are the following two clauses in the knowledge base:

$$C_1 = a \vee b \vee \neg l$$

$$C_2 = b \vee c \vee l$$

Their merge resolvent is

$$C_3 = a \vee b \vee c$$

Now if the unit clauses $\neg a$ and $\neg c$ are given as premises, from the three clauses C_1, C_2 and C_3 by unit resolution one can derive b. Using the same premises with only C_1 and C_2, unit resolution yields the two clauses $b \vee \neg l$ and $b \vee l$, which cannot be resolved by unit resolution, hence b is not derived by unit resolution in this case. Therefore, the clause C_3 is not redundant with respect to unit resolution, and must be kept in the compilation of the knowledge base.

del Val [16] gives cases in which unit-resolution-complete compilation can discard an exponential number of prime implicates, and gives some favorable experimental comparisons with prime implicates methods for some benchmark problems from theorem proving. He also shows that adapting Tison's algorithm [55] for finding prime implicates one can dramatically reduce the number of non-redundant prime implicates, and hence the size of the compiled knowledge base. Hence the choice of a good algorithm for generating prime implicates can be crucial for the feasibility of compilation.

A limit of del Val's method is that it can be applied only to CNF knowledge bases, since both the filtering condition on prime implicates and the deduction in the compiled knowledge base are based on resolution.

3.3. Theory prime implicates

Probably the most refined method at present for exact compilation based on prime implicates is the one proposed by Marquis [39]. He starts observing that prime-implicants and prime-implicates methods are based on transforming the problem $KB \models Q$ (Q being a clause), involving the entire knowledge base, into *local* tests involving one prime implicant/implicate at a time. He proposes to enhance such local tests with a theory, while keeping its complexity to be polynomial-time.

Definition 4 (cf. [39]) *A* theory prime implicate *of a knowledge base KB with respect to a theory Φ is a clause C such that $KB \cup \Phi \models C$ and for each other clause C' if $KB \cup \Phi \models C'$ and $\Phi \cup C' \models C$ then also $\Phi \cup C \models C'$.*

Note that when Φ is an empty knowledge base one obtains the definition of prime implicate. Hence theory prime implicates extend prime implicates. Also the set containment test of prime implicates is extended: given a theory Φ, for every CNF query Q, $KB \models Q$ iff for every clause C' of Q there is a theory prime implicate C of KB with respect to Φ such that $\Phi \cup \{C\} \models C'$. Again, taking an empty Φ the usual property for prime implicates is obtained.

Observe that checking $\Phi \cup \{C\} \models C'$ is equivalent to check, for each literal $l_i \in C$, whether $\Phi \models \neg l_i \vee C'$. The key point is that *if deduction in the theory Φ can be performed in polynomial-time* (e.g., if Φ is a Horn theory) entailment of CNF queries can be computed in time polynomial in the size of the set of theory prime implicates. Under the same condition, also the check for primality can be done in polynomial time.

Marquis suggests as good candidates for Φ the set of all Horn clauses of KB, the set of binary (Krom) clauses of KB, and many others (in general, any subset of KB such that entailment is tractable). He also gives examples in which the number of theory prime implicates is exponentially smaller than the number of del Val's filtered prime implicates, and some experimental favorable comparisons on benchmark problems with del Val's method.

In [40] Marquis and Sadaoui give an algorithm for computing theory prime implicates, which is a modification of the prime-implicants compilation algorithm of Madre and Coudert [38] based on Binary Decision Diagrams. With this algorithm, the initial knowledge base does not need to be in CNF, and the prime implicates of KB need not to be generated; this reduces considerably compilation operations, as shown by some experiments.

4. Approximate compilation

As we saw in Section 2.1, it is not always possible to make on-line reasoning more efficient in all cases, as theoretical results tell us that some basic problems in KR which are polynomially intractable (e.g., logical entailment of clauses in propositional logic) are most likely not compilable.

As we mentioned in Section 1, in KR two other different approaches have been taken for addressing computational hardness of reasoning problems: language restriction and theory approximation. Such approaches have traditionally been used for achieving tractability in reasoning problems in which there is no distinction between on-line and off-line part. In the setting of knowledge compilation, they can be described as follows:

Language restriction. We can restrict the language used for representing knowledge (either the fixed or the varying part) so that the formalization of interesting cases is still possible, but resulting problems are compilable.

Theory approximation. We can give up either soundness or completeness in the answer to the original reasoning problem we want to solve. This could either mean to do off-line reasoning in such a way that information is lost, or to preserve information and lose either soundness or completeness during the on-line reasoning part. In both cases, again the goal is to have fast on-line reasoning.

The language restriction approach has not been considered very much in the knowledge compilation setting. On the other hand, the theory approximation approach has had some success; we introduce it with an analogy.

In optimization problems, approximate solutions are quite common and meaningful. As an example, in the approximate version of the "Bin-packing" optimization problem we do not to look for the minimum number of bins, but rather we look for a "good" number – i.e., not "too far" from the optimal one.

However, in KR we are not dealing with numerical problems, so there is not an obvious metrics that tells us "how far we are" from the right answer to a logical entailment problem. Therefore the very notion of approximation of the answer to an entailment problem should be grounded on a semantical basis.

In approximate knowledge compilation the central idea is that answers to a query using the compiled knowledge base can be either unsound or incomplete (but not both). Intuitively, a sound-but-incomplete answer approximates from one side the correct answer (a "yes" is correct while a "no" is in fact a "don't know") and a complete-but-unsound answer approximates the correct one from the other side (a "no" is correct, while a "yes" is a "don't know"). Obviously, also in this case one aims at using less computational resources than in the sound-and-complete case. One could use the approximation to look for a quick answer, and in case it is a "don't know", consult a slower sound-and-complete procedure.

An approximation A of a knowledge base KB is sound when for every query Q, if $A \models Q$ then $KB \models Q$. In this case A is called an *upper bound* (UB) for KB. Observe that A is an UB for KB iff $KB \models A$ holds. Dually, an approximation B of a knowledge base KB is complete when for every query Q, if $B \not\models Q$ then $KB \not\models Q$. In this case, B is called a *lower bound* (LB) for KB, and $B \models KB$ holds.

4.1. Anytime versions of exact methods

We first note that any of the methods for exact compilation shown in Section 3 can be stopped before it is completed, obtainining an approximation of the knowledge base. More precisely, methods computing implicants like Schrag's [49] yield lower bounds: if a query Q is not implied by one of the already computed implicants of KB, then $KB \not\models Q$. On the other hand, all other methods based on implicates, as del Val's [16] and Marquis' [39], yield upper bounds if stopped before they completed compilation: if one of the already computed implicates implies the query Q, then $KB \models Q$.

4.2. Horn approximations

An original method for approximate knowledge compilation has been proposed by Selman and Kautz in [51] in the context of propositional knowledge bases, and extended to other languages in [52]. The basic idea of their method is to compile a knowledge base into a formula which belongs to a syntactic class which guarantees polynomial-time inference. As an example, a propositional formula Σ can be compiled into a Horn one Σ', since $\Sigma' \models \gamma$, where γ is a clause, can be answered in time $O(|\Sigma'| + |\gamma|)$.

There exist two different ways of doing such a compilation. In the first case the compiled formula satisfies the relation $\Sigma' \models \Sigma$, or equivalently $\mathcal{M}(\Sigma') \subseteq \mathcal{M}(\Sigma)$ – where $\mathcal{M}(\Phi)$ denotes the set of models of the formula Φ. For this reason Σ' is called a *Horn* LB of Σ. As an example – taken from [51] – let Φ be the knowledge base

$$(man \longrightarrow person)$$
$$(woman \longrightarrow person)$$
$$(man \vee woman)$$

The formula $\Phi_{lb} = man \wedge woman \wedge person$ is a Horn LB of Φ. The second form of compilation is dual. The compiled version of Σ is a Horn formula Σ' that satisfies the relation $\Sigma \models \Sigma'$, or equivalently $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma')$. Σ' is called a *Horn* UB of Σ. Returning to the previous example, the formula $\Phi_{ub} = (man \longrightarrow person) \wedge (woman \longrightarrow person)$ is a Horn UB of Φ.

Selman and Kautz notice that some LBs are better than others. In the previous example, both $\Phi_{lb1} = man \wedge woman \wedge person$ and $\Phi_{lb2} = man \wedge person$ are Horn LBs of Φ. Φ_{lb2} seems to be a better approximation than Φ_{lb1}, since $\mathcal{M}(\Phi_{lb1}) \subset \mathcal{M}(\Phi_{lb2}) \subset \mathcal{M}(\Phi)$, hence Φ_{lb2} is in some precise sense "closer" to Φ than Φ_{lb1}. This consideration leads to the notion of a *Horn greatest lower bound* (GLB) of a formula Σ, which is a Horn formula Σ_{glb} such that $\mathcal{M}(\Sigma_{glb}) \subseteq \mathcal{M}(\Sigma)$ and for no Horn formula Σ' it holds that $\mathcal{M}(\Sigma_{glb}) \subset \mathcal{M}(\Sigma') \subseteq \mathcal{M}(\Sigma)$. In the previous example Φ_{lb2} is a Horn GLB of Φ.

The same argument can be done for Horn upper bounds: in our example both $\Phi_{ub1} = (man \longrightarrow person) \wedge (woman \longrightarrow person)$ and $\Phi_{ub2} = person$ are Horn UBs of Φ, but $\mathcal{M}(\Phi) \subset \mathcal{M}(\Phi_{ub2}) \subset \mathcal{M}(\Phi_{ub1})$, hence Φ_{ub2} is a better approximation of Φ. A *Horn least upper bound* (LUB) of a formula Σ is a Horn formula Σ_{lub} such that $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma_{lub})$ and for no Horn formula Σ' it holds that $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma') \subset \mathcal{M}(\Sigma_{lub})$. Φ_{ub2} is a Horn LUB of Φ.

Selman and Kautz's proposal is to approximate inference with respect to a propositional formula Σ by using its Horn GLBs and LUBs. In this way inference could be unsound or incomplete, but it is anyway possible to spend more time and use a general inference procedure to determine the answer directly from the original formula. The general inference procedure could still use the approximations to prune its search space.

This basic idea has been extended to other polynomial-time target languages in [52, 17], some of which non-propositional. Other researchers have investigated computational properties of Horn compilations [6, 28].

5. Compilation of non-monotonic knowledge bases

As we already said, Definition 1 sets two basic requirements, by means of the two polynomials it refers to. While polynomiality of the size of the output of the preprocessing part is mandatory (i.e., we cannot give up polynomial p_1), we might consider doing on-line reasoning that is more than just polynomial (i.e., relax the requirement that algorithm ASK must be polynomial-time). As an example, it could make sense to preprocess the fixed part of a PSPACE-complete problem, hoping that the residual on-line problem is "just" NP-complete.

As a matter of fact, this question has been widely investigated, especially in the non-monotonic reasoning literature, where we can find several attempts to "translate" a formula of some non-monotonic logic into a formula of classical logic. The relation to compilation is evident, since entailment problems in most propositional non-monotonic logics are either Σ_2^p- or Π_2^p-complete [11], while reasoning problems in classical propositional logic are typically either NP- or coNP-complete.

When we study the subject from this general point of view, we get to conclusions similar to those relative to the notion of compilation of Definition 1: It is very unlikely that every non-monotonic formula can be always translated into an "equivalent" propositional formula of polynomial size.

In this section we consider three non-monotonic formalisms: circumscription in Section 5.1, default logic in Section 5.2, and belief revision in Section 5.3.

5.1. Circumscription

The *circumscription* $CIRC(T)$ [42, 37] of a propositional formula T can be defined through the notion of *minimal* model, i.e., a model of T in which the set of letters which are assigned true is minimal with respect to set containment. This simple definition can be enriched in several ways. The problem of checking whether $CIRC(T) \models \gamma$, γ being a clause, is Π_2^p-complete [23].

As for the possibility to translate $CIRC(T)$ into an equivalent formula T' of size polynomial in $|T|$, in [9] it is shown that this would imply the collapse of the polynomial hierarchy. Several definitions of equivalence are taken into account, and each of them has different implications. In particular, if T' has exactly the same set of *models* as $CIRC(T)$, then a polynomial-size T' exists iff NP $\subseteq$ NC^1/poly, which implies that the polynomial hierarchy collapses at the second level (Σ_2^p= PH). If T' has exactly the same set of *theorems* as $CIRC(T)$, then a polynomial-size T' exists iff NP $\subseteq$ coNP/poly, which implies that the polynomial hierarchy collapses at the third level (Σ_3^p=PH, which is implied by Σ_2^p= PH).

The paper by Gelfond et al. [26] can be considered the first paper showing a compilation of circumscriptive knowledge bases into propositional logic. They show that a knowledge base KB interpreted under circumscription can be expressed into one interpreted in propositional logic by adding to KB negative clauses that are true in all minimal models. They call the result the Extended Generalized Closed World Assumption (EGCWA) of KB. More formally,

$$\begin{aligned} CIRC(T) &= EGCWA(T) \\ &= T \cup \{\gamma | \gamma \text{ is a negative clause and } \gamma \\ &\qquad \text{is true in all minimal models of } T\} \end{aligned}$$

Of course, from a knowledge compilation point of view only the minimal clauses should be added. In [9] worst cases are given, for which such a compilation outputs an exponential number of minimal clauses.

Nerode et al. [44] compile a database interpreted under circumscription by finding its minimal models using a mixed integer linear programming. Then they compute two subsets of the minimal models (the minimal and maximal ones with respect to set inclusion) from which answers to positive and negative queries can be computed. Since they answer queries directly from the computed models, they compile circumscription directly into a polynomial-time data structure (the database of minimal models). Both parallel and prioritized circumscription are considered.

Another field of investigation related to knowledge compilation is the search for *collapsible* cases of circumscription. Circumscription of a first-order formula is a second-order formula, which in general is not collapsible to a first-order one [37]. Many researchers tried to find classes of first-order formulae whose circumscription is equivalent to a first-order formula, and proposed methods for finding such equivalent formulae. Some of such methods (cf. e.g., [19]) work for the propositional case, and can therefore be used as methods for knowledge compilation.

5.2. Default logic

A propositional *default theory* [47] is a pair $\langle D, W\rangle$, where D is a collection of special inference rules called *defaults*, and W is a set of propositional formulae. The semantics of a default theory $\langle D, W\rangle$ is based on the notion of *extension*, which is a possible state of the world according to the knowledge base. Intuitively, an extension is a deductively closed set of formulae in which a maximal set of default rules have been applied (usually applying all default rules leads to inconsistency, therefore some have to be dropped). A default theory can have zero, one, or several extensions.

A typical inference problem is *skeptical default reasoning*, i.e., prove that a formula follows from all extensions of $\langle D, W\rangle$. Such a problem is Π_2^p-complete [29]. As for the possibility of compiling $\langle D, W\rangle$ into a propositional formula of size polynomial in $|D| + |W|$, limitations similar to those of circumscription are proved in [8, 27].

Ben-Eliyahu and Dechter [3] show how to compile a default theory into a propositional formula, such that default extensions are one-to-one with models of the propositional formula. They also point out a particular class of default theories for which the compilation takes polynomial time (hence polynomial space too), proving that for this class of theories credulous default reasoning is NP-complete while skeptical is coNP-complete. An application of their method in the context of disjunctive logic programs can be found in [2].

5.3. Belief revision

Belief revision has to do with evolution of the state of a knowledge base: How does our set of beliefs change when new information arrives? Several researchers attempted to define formalizations of belief revision, subject to the following question. Suppose we have a set of beliefs, represented by a propositional theory T; at some point new information arrives, let's say a propositional formula P. If T and P are inconsistent, can we still infer something reasonable from all the knowledge we have? In other words, what are the logical consequences of the theory T when it is revised by the formula P?

Many formalizations for belief revision have been proposed in the literature [22]. Starting from a propositional knowledge base, a reasonable requirement is that after a revision the knowledge base is still expressed as a propositional formula. However, for most of the belief revision formalisms query answering in a revised knowledge base is above coNP. Hence translating a revised knowledge base into a propositional formula again can be seen as a compilation task.

Winslett [56] proposes a formalization of belief revision and describes a method for compiling a revised knowledge base by using new sets of propositional variables. She shows that when the size of the revising formula is bounded by a constant, the size of the result is polynomial.

Cadoli et al. [7] analyze many formalizations for belief revision, pointing out which ones admit a polynomial-size compilation and which ones are unlikely to have any. They also study iterated belief revision, and show that any compilation method for Winslett's belief revision is very likely to be worst-case exponential in the number of revisions.

6. Related work

In this section we briefly mention other forms of knowledge compilation, namely compilation of knowledge bases which already support polynomial-time inference algorithms (Section 6.1), and compilation that does not yield knowledge bases (Section 6.2). As for compilation of first-order knowledge bases, a preliminary investigation is shown in [18]. Finally, we briefly compare knowledge compilation with the related idea of *fixed parameter tractability* (Section 6.3).

6.1. Optimizing polynomial-time query answering

Compilation can make sense also for knowledge bases for which the problem of query answering is already doable in polynomial-time. For Horn knowledge bases, Hammer and Kogan [31] propose a method for optimizing the size of a Horn knowledge base. They prove that finding a minimal-size equivalent Horn knowledge base is an NP-complete problem, and provide a quadratic-time algorithm [30] that finds an equivalent Horn knowledge base whose size approximates the minimal one.

6.2. Non-logical compilations

The compilation methods presented throughout this paper always output a logical formula. However, the compilation can in general output any data structure that helps on-line query answering. In particular, when the number of possible queries is polynomial, a database of precomputed answers is always theoretically feasible.

Moses and Tennenholtz [43] consider query languages whose queries can be decomposed as a conjunction of basis queries. When the number of basis queries is polynomial in the number of variables of the query language, they call it an *efficient basis*. For queries admitting an efficient basis, one can compute off-line and store the answer to basis queries, and obtain on-line the answer to any other query by table look-up and conjunction of answers. They observe that queries in k-CNF admit an efficient basis, namely all clauses with k literals.

Along the same lines, Cadoli et al. [8] observed that for k-CNF queries compilation of circumscriptive knowledge bases into polynomial-size structures is possible. They also point out restricted forms of default theories and queries for which compilation to a polynomial-size structure is possible.

Khardon and Roth [35] propose a sophisticated way of representing the models of a knowledge base, which stems from results on learning monotone boolean functions. They consider sets of assignments to a CNF formula F such that for each clause in F, there is at least one assignment falsifying the clause. They call such a set a *basis* for F. Each basis assignment b defines an ordering on all assignments, which is a lattice over $\{\mathsf{false}, \mathsf{true}\}^n$ with $\mathsf{false} < \mathsf{true}$ if b assigns true to the i-th variable, and $\mathsf{true} < \mathsf{false}$ otherwise. Then they define the set of *characteristic models* of a knowledge base KB with respect to a basis, which is the union of all sets of miminal models of KB according to assignments in the basis.

Particular classes of queries, such as Horn CNF, reverse-Horn CNF (clauses with at most one negative literal) and $\log n$-CNF (clauses with at most $\log n$ literals, where n is the total number of variables) have a polynomial-size basis, and the set of characteristic models of a knowledge base with respect to such a basis is sufficient to answer all queries in the class. Observe that the number of $\log n$-CNF is superpolynomial in n, hence this method is strictly more general than the compilation proposed by Moses and Tennenholtz ($\log n$-CNF queries do not admit an efficient basis). A related approach has been recently proposed in [5].

6.3. Fixed-parameter tractability

Another body of research bearing similarities with the idea of knowledge compilation is "fixed parameter tractability" [20, 21]. There are problems such that the input can be naturally split into two parts, e.g., whether a graph G has a clique of size k or not. In this case k is a parameter that can be fixed. Roughly speaking, a problem is fixed parameter tractable if it can be solved in time $f(k) \cdot n^c$, where n is the size of the input, c is a constant, and f is a function independent of the input. Although both in knowledge compilation and in fixed parameter tractability the input to a problem is split into two parts, and in both cases the aim is to prove that some NP-hard problems become tractable if one part of the input is treated in some special way, there are some major differences between the fixed parameter tractability of a problem and the feasibility of its off-line processing.

In a nutshell, knowledge compilation assumes that there are several instances of the same problem, all sharing a fixed part, and it is not interested in finding an expression of the complexity in terms of both parts of the input. Moreover, in knowledge compilation the knowledge base is known well before the varying part, while fixed parameter tractability is defined for a single instance, and the algorithm comes to know both parts of the input at the same time. An extensive and formal analysis of the similarities and differences of the two approaches is performed in [10].

7. Conclusions

In this paper we surveyed knowledge compilation, a technique for making reasoning computationally easier by exploiting the output of a preprocessing phase. The first proposals of such a technique concerned prime implicants/implicates. Since then, knowledge compilation has become very popular in KR, and it has been applied to propositional as well as non-monotonic knowledge bases, and also to belief revision problems.

Evaluation of the performances of the technique, obtained by timing algorithms for reasoning in the compiled knowledge bases, show that knowledge compilation is mature for applications. Recently there has also been some interest on theoretical limits of knowledge compilation, and relations to complexity classes have been shown.

Investigation on compilation of first-order knowledge bases has just started, and will probably receive increasing attention in the future.

Acknowledgments

The authors are grateful to Pierre Marquis and to two anonymous referees for their very useful suggestions, which helped improving the readability of the paper. This work has been supported by ASI (Italian Space Agency), MURST (Italian Ministry for University and Scientific and Technological Research) and CNR (Italian Research Council).

References

[1] O. L. Astrachan and M. E. Stickel. Caching and lemmaizing in model elimination theorem provers. In *Proceedings of the Eleventh International Conference on Automated Deduction* (*CADE-92*), pages 224–238, 1992.

[2] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

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

[4] A. Borgida, R. J. Brachman, D. W. Etherington, and H. A. Kautz. Vivid knowledge and tractable reasoning: Preliminary report. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (*IJCAI-89*), pages 1146–1152, 1989.

[5] Y. Boufkhad, É. Grégoire, P. Marquis, B. Mazure, and L. Saïs. Tractable cover compilations. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (*IJCAI-97*), pages 122–127, 1997.

[6] M. Cadoli. Semantical and computational aspects of Horn approximations. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (*IJCAI-93*), pages 39–44, 1993.

[7] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems* (*PODS-95*), pages 151–162, 1995. Extended version available as Technical Report DIS 34-96, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", November 1996.

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

[9] M. Cadoli, F. M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.

[10] 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.

[11] M. Cadoli and M. Schaerf. A survey of complexity results for non-monotonic logics. *Journal of Logic Programming*, 17:127–160, 1993.

[12] T. Castell and M. Cayrol. Computation of prime implicates and prime implicants by the Davis and Putnam procedure. In *Workshop on "Advances in Propositional Deduction", held at ECAI-96*, pages 61–64, 1996.

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

[14] J. de Kleer. An improved incremental algorithm for generating prime implicates. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (*AAAI-92*), pages 780–785, 1992.

[15] R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning* (*KR-94*), pages 134–145, 1994.

[16] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning* (*KR-94*), pages 551–561, 1994.

[17] A. del Val. An analysis of approximate knowledge compilation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI-95*), pages 830–836, 1995.

[18] A. del Val. Approximate knowledge compilation: The first order case. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (*AAAI-96*), pages 498–503, 1996.

[19] P. Doherty, W. Lukaszewicz, and A. Szalas. Computing circumscription revisited: A reduction algorithm. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI-95*), pages 1502–1508, 1995.

[20] R. G. Downey and M. F. Fellows. Fixed-parameter tractability and completeness: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[21] R. G. Downey and M. F. Fellows. Parameterized complexity, February 1995. Manuscript, 411 pages. To appear by Springer in 1997.

[22] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and conterfactuals. *Artificial Intelligence Journal*, 57:227–270, 1992.

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

[24] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, second edition, 1996.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, Ca, 1979.

[26] M. Gelfond, H. Przymusinska, and T. Przymusinsky. On the relationship between circumscription and negation as failure. *Artificial Intelligence Journal*, 38:49–73, 1989.

[27] 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.

[28] G. Gogic, C. Papadimitriou, and M. Sideri. Incremental recompilation of knowledge. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (*AAAI-94*), pages 922–927, 1994.

[29] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[30] P. L. Hammer and A. Kogan. Horn functions and their DNFs. *Information Processing Letters*, 44:23–29, 1992.

[31] P. L. Hammer and A. Kogan. Optimal compression of propositional Horn knowledge bases: Complexity and approximation. *Artificial Intelligence Journal*, 64:131–145, 1993.

[32] T. Hogg, B. A. Hubermann, and C. P. Williams (eds.). Special volume on frontiers of problem solving: Phase transitions and complexity. *Artificial Intelligence Journal*, 81, 1996.

[33] P. Jackson and J. Pais. Computing prime implicants. In *Proceedings of the Tenth International Conference on Automated Deduction* (*CADE-90*), pages 543–557, 1990.

[34] 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.

[35] R. Khardon and D. Roth. Reasoning with models. *Artificial Intelligence Journal*, 87:187–213, 1996.

[36] H. J. Levesque. Making believers out of computers. *Artificial Intelligence Journal*, 30:81–108, 1986.

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

[38] J.C. Madre and O. Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (*IJCAI-91*), pages 294–299, 1991.

[39] P. Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI-95*), pages 837–843, 1995.

[40] P. Marquis and S. Sadaoui. A new algorithm for computing theory prime implicates compilations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (*AAAI-96*), pages 504–509, 1996.

[41] P. Mathieu and J.-P. Delahaye. A kind of logical compilation for knowledge bases. *Theoretical Computer Science*, 131:197–218, 1994.

[42] J. McCarthy. Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence Journal*, 13:27–39, 1980.

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

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

[45] T.-H. Ngair. A new algorithm for incremental prime implicate generation. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (*IJCAI-93*), pages 46–51, 1993.

[46] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publishing Company, 1994.

[47] R. Reiter. A logic for default reasoning. *Artificial Intelligence Journal*, 13:81–132, 1980.

[48] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenace systems: Preliminary report. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (*AAAI-87*), pages 183–188, 1987.

[49] R. Schrag. Compilation for critically constrained knowledge bases. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (*AAAI-96*), pages 510–515, 1996.

[50] R. Schrag and J. Crawford. Implicates and prime implicates in random 3SAT. *Artificial Intelligence Journal*, 81:199–222, 1996.

[51] B. Selman and H. A. Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (*AAAI-91*), pages 904–909, 1991.

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

[53] B. Selman, D.G. Mitchell, and H.J. Levesque. Generating

hard satisfiability problems. *Artificial Intelligence Journal*, 81:17–30, 1996.

[54] J. R. Slagle, C. L. Chang, and R. C. T. Lee. A new algorithm for generating prime implicants. *IEEE Transactions on Computers*, 19(4):304–310, 1970.

[55] P. Tison. Generalized consensus theory and application to the minimization of boolean functions. *IEEE transactions on electronic computers*, 4:446–456, 1967.

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