A General Framework for Knowledge Compilation

Henry Kautz and Bart Selman

AI Principles Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974 USA
kautz@research.att.com
selman@research.att.com

Abstract

Computational efficiency is a central concern in the design of knowledge representation systems. In order to obtain efficient systems it has been suggested that one should limit the form of the statements in the knowledge base or use an incomplete inference mechanism. The former approach is often too restrictive for practical applications, whereas the latter leads to uncertainty about exactly what can and cannot be inferred from the knowledge base. We present a third alternative, in which knowledge given in a general representation language is translated (compiled) into a tractable form — allowing for efficient subsequent query answering.

We show how propositional logical theories can be compiled into Horn theories that approximate the original information. The approximations bound the original theory from below and above in terms of logical strength. The procedures are extended to other tractable languages (for example, binary clauses) and to the first-order case. Finally, we demonstrate the generality of our approach by compiling concept descriptions in a general frame-based language into a tractable form.

This paper appears in the *Proceedings of the International Workshop on Processing Declarative Knowledge*, Kaiserlautern, Germany, July 1991. Book version of the proceedings forthcoming from Springer-Verlag.

1 Introduction

A striking feature of commonsense reasoning is that it is *fast*. People can quickly and effortlessly perform such tasks as planning a trip, interpreting a story, or answering a question without relying solely on memory. Levesque [89] gives the example of the question, "Could a crocodile run a steeplechase?" You can answer the question immediately, even though you have never thought about it before! The "logicist" approach to artificial intelligence represents commonsense knowledge by logical formulas, and views reasoning as a kind of formal theorem-proving. This accounts for the flexibility of commonsense reasoning, but not its speed. In both theory and practice the complexity of theorem-proving with general logical theories is very high—exponential or worse.

One way to make the logicist approach more computationally attractive is to restrict the expressive power of the representation language, so that fast, special-purpose inference algorithms can be employed. But this usually renders the language too limited for practical application [Doyle and Patil, 1991], and leaves unanswered the question of what to do with information that *cannot* be represented in the restricted form.

This paper describes an approach to efficient symbolic inference called *knowledge compilation*, which overcomes these objections. We allow the knowledge base to be specified in a general, unrestricted representation language. The system then computes *approximations* to the knowledge base in a restricted and efficient language. We show how the approximations can be used to speed up inference *without* giving up correctness or completeness: computational costs are simply shifted from "run time" question-answering to the "off-line" compilation process.

The paper begins with an example of approximating general propositional theories by Horn theories. Next we describe the general knowledge compilation framework, and examine a number of other knowledge compilation systems, including ones involving generalizations of Horn clauses, first-order theories, logic programs, and terminological logics.

2 Compiling Propositional Theories

This section briefly introduces the knowledge compilation approach using the concrete example of approximating general propositional theories by Horn theories. This form of knowledge compilation was first introduced in [Selman and Kautz, 1991a]. That paper gives a much more detailed account; the reader who is familiar with it can proceed to the next section.

2.1 Definitions

We assume a standard propositional language, and use p, q, r, and s to denote propositional letters and x, y, and z to denote literals (a *literal* is either a propositional letter, called a positive literal, or its negation, called a negative literal). A *clause* is a disjunction of literals, and can be represented by the set of literals it contains. A clause is *Horn* if and only if it contains at most one positive literal; a set of such clauses is called a *Horn theory*. Formulas are given in conjunctive normal form (a conjunction of disjuncts), so they can be represented by a set of clauses.

In general, determining whether a given formula (the *query*) follows from a set of formulas in a knowledge base is intractable (provided $P \neq NP$) [Cook, 1971]. However, when the knowledge base contains only Horn clauses the problem can be solved in linear time [Dowling and Gallier, 1984]. We therefore take as the goal of our knowledge compilation process the translation of an arbitrary set of clauses into a logically equivalent set of Horn clauses. Since an exact translation is often not possible, we use two sets of Horn clauses to approximate the original theory. The basic idea is to bound the set of models (satisfying truth assignments) of the original theory from below and from above by Horn theories. In the following definition, $\mathcal{M}(\Sigma)$ denotes the set of satisfying truth assignments of the theory Σ.

Definition: Horn lower-bound and Horn upper-bound
Let Σ be a set of clauses. The sets Σ_{lb} and Σ_{ub} of Horn clauses are respectively a Horn lower-bound and a Horn upper-bound of Σ iff

$$\mathcal{M}(\Sigma_{\text{lb}}) \subseteq \mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma_{\text{ub}})$$

or, equivalently,

$$\Sigma_{\text{lb}} \models \Sigma \models \Sigma_{\text{ub}}$$

Note that the bounds are defined in terms of models. The reader is cautioned not to associate "lower" with "logically weaker": The *lower* bound has fewer models than the original theory, and is thus logically *stronger* than (*i.e.*, implies) the original theory; whereas the *upper* bound has more models and is thus logically *weaker* than (*i.e.*, is implied by) the original theory.

Instead of simply using any pair of bounds to characterize the initial theory, we wish to use the best possible ones: a *greatest* Horn lower-bound and a *least* Horn upper-bound.

Definition: Greatest Horn lower-bound (GLB)
Let Σ be a set of clauses. The set Σ_{glb} of Horn clauses is a greatest Horn lower-bound of Σ iff $\Sigma_{\text{glb}} \models \Sigma$ and there is no set Σ' of Horn clauses such that $\Sigma_{\text{glb}} \models \Sigma' \models \Sigma$ and $\Sigma \not\models \Sigma'$.

Definition: Least Horn upper-bound (LUB)
Let Σ be a set of clauses. The set Σ_{lub} of Horn clauses is a least Horn upper-bound of Σ iff $\Sigma \models \Sigma_{\text{lub}}$ and there is no set Σ' of Horn clauses such that $\Sigma \models \Sigma' \models \Sigma_{\text{lub}}$ and $\Sigma' \not\models \Sigma$.

We call these bounds *Horn approximations* of the original theory Σ. The LUB is unique (up to logical equivalence) because the conjunction of any two upper bounds is another, possibly smaller upper-bound.

Example: Consider the non-Horn theory $\Sigma = (\neg p \vee r) \wedge (\neg q \vee r) \wedge (p \vee q)$. $p \wedge q \wedge r$ is an example of a Horn lower-bound; both $p \wedge r$ and $q \wedge r$ are GLBs; $(\neg p \vee r) \wedge (\neg q \vee r)$ is an example of a Horn upper-bound; and r is the LUB. The reader can verify these bounds by noting that

$$(p \wedge q \wedge r) \models (p \wedge r) \models \Sigma \models r \models ((\neg p \vee r) \wedge (\neg q \vee r))$$

Moreover, there is no Horn theory Σ' different from $p \wedge r$ such that $(p \wedge r) \models \Sigma' \models \Sigma$. Similar properties hold of the other GLB and of the LUB.

2.2 Using Bounds for Fast Inference

Let us now consider how these approximations can be used to improve the efficiency of a knowledge representation system. Suppose a knowledge base (KB) contains the set of clauses Σ, and we want to determine whether the formula α is implied by the KB. The system can proceed as follows. First, it tries to obtain an answer quickly by using the Horn approximations. If $\Sigma_{\mathrm{lub}} \models \alpha$ then it returns "yes, logically follows" or if $\Sigma_{\mathrm{glb}} \not\models \alpha$ then it returns "no, does not logically follow." So far, the procedure takes only time linear in the length of the approximations. (We assume that the lengths of the Horn approximations are roughly the same as that of the original theory; we return to this issue below.) In case no answer is obtained, the system could simply return "don't know," or it could decide to spend more time and use a general inference procedure to determine the answer directly from the original theory.[2] Thus the system can answer certain queries in linear time, resulting in a improvement in its overall response time. Exactly how many queries can be handled directly by the Horn approximations depends on how well the bounds characterize the original theory. Note that we give up neither soundness nor completeness, because we can always fall back to the original theory.

To make this discussion more concrete, consider the following interpretation of the propositional letters in the previous example:

$p \equiv$ Sally is a doctor
$q \equiv$ Sally is a lawyer
$r \equiv$ Sally is rich
$s \equiv$ Sally is a vegetarian

The theory Σ can be understood as asserting that if Sally is a doctor, then she is rich; if Sally is a lawyer, then she is rich; and Sally is a doctor or a lawyer. The LUB asserts that Sally is rich (r), and suppose the system generates the GLB that asserts that Sally is a rich doctor ($p \wedge r$). Now, the query "Does it follow that Sally is rich?" can immediately be answered "yes" because r is implied by the LUB. The query "Does it follow that Sally is a doctor?" is answered "don't know" because a is not implied by the LUB; the fact that it follows from the GLB is not relevant. Finally, the query "Does it follow that Sally is a vegetarian?" is answered "no", because s is not implied by the GLB.

Note these questions can all be answered in time linear in the size of the upper and lower bounds; thus, even if Σ were expanded to contain many more complex domains axioms the queries could still be answered efficiently. Furthermore, the GLB is always no larger than the original theory Σ itself. However, one can construct theories which have an exponentially larger least Horn upper-bound. We are currently investigating the question of whether it is always possible to generate a compact encoding of the LUB (one polynomial in the size of Σ) by introducing new propositional letters.

The system as described does not make any closed world assumptions; for example, it does not conclude "Sally is not a vegetarian." There may be a connection, however, between knowledge compilation and closed world reasoning, in that the logical closure of a theory is a lower bound of the theory.[3]

[2] The general inference procedure could still use the approximations to prune its search space.
[3] This observation is due to Mukesh Dalal and David Etherington.

GLB Algorithm
Input: a set of clauses $\Sigma = \{C_1, C_2, \ldots, C_n\}$.
Output: a greatest Horn lower-bound of Σ.
begin
  $L :=$ the lexicographically first Horn–strengthening of Σ
  loop
    $L' :=$ lexicographically next Horn–strengthening of Σ
    if none exists **then exit**
    if $L \models L'$ **then** $L := L'$
  end loop
  remove subsumed clauses from L
  return L
end

Figure 1: Algorithm for generating a greatest Horn lower-bound.

2.3 Computing Horn Approximations

We now turn to the problem of generating Horn approximations. As shown in [Selman and Kautz, 1991a], there does not exist a polynomial time procedure for generating such approximations (provided P$\neq$NP). Computing the Horn approximations is therefore treated as a compilation process in which the computational cost is amortized over the total set of subsequent queries to the KB. Since the approximations may be needed for query answering before the compilation process finishes, it is desirable to employ procedures that can output lower- and upper-bounds as intermediate results, generating better and better bounds over time. That is, the approximation algorithms should be "anytime" procedures [Boddy and Dean, 1988].

Central to our algorithm for computing the GLB (Figure 1) is the following notion.

Definition: Horn-strengthening
A Horn clause C_{H} is a Horn-strengthening of a clause $C = \{x_1, \ldots, x_n\}$ iff $C_{\mathrm{H}} \subseteq C$ and there is no Horn clause C'_{H} such that $C_{\mathrm{H}} \subset C'_{\mathrm{H}} \subseteq C$.

The **GLB algorithm** systematically searches through the various possible Horn-strengthenings of the clauses of the original theory, looking for a most general one. This approach is illustrated with the following example.

Example: Consider the theory $(\neg p \vee q) \wedge (p \vee q \vee r)$. The **GLB algorithm** first tries the Horn-strengthening $L = (\neg p \vee q) \wedge p$, and then $L' = (\neg p \vee q) \wedge q$. Since $L \models L'$, L is set to L', and the algorithm proceeds. (L is used to store best bound obtained so far.) Since the last Horn strengthening $(\neg p \vee q) \wedge r$ is not an improvement, the algorithm returns q as a GLB (the clause $(\neg p \vee q)$ is removed because it is subsumed by q).

Note that the **GLB algorithm** is indeed an anytime algorithm: L represents some lower-bound whenever the algorithm is interrupted.

```
LUB Algorithm
Input: a set of clauses Σ = Σ_H ∪ Σ_N, where
        Σ_H is a set of Horn clauses, and Σ_N is a
        set of non-Horn clauses.
Output: a least Horn upper-bound of Σ.
begin
    loop
        try to choose clause C_0 ∈ Σ_H ∪ Σ_N and
            C_1 ∈ Σ_N, such that C_2 = Resolve(C_0, C_1)
            is not subsumed by any clause in Σ_H ∪ Σ_N
        if no such choice is possible then exit loop
        if C_2 is Horn then
          delete from Σ_H and Σ_N any clauses
                 subsumed by C_2
          Σ_H := Σ_H ∪ {C_2}
        else
          delete from Σ_N any clauses subsumed by C_2
          Σ_N := Σ_N ∪ {C_2}
        end if
    end loop
    return Σ_H
end
```

Figure 2: Algorithm for generating a least Horn upper-bound.

The **LUB algorithm** shown in Figure 2 exploits that fact that the LUB is logically equivalent to the set of Horn resolvents of the theory. Since even a Horn theory can have exponentially many resolvents (all Horn), it is very inefficient to simply generate all resolvents of the original theory while collecting the Horn ones. It sufficient, however, to resolve only pairs of clauses containing at least one non-Horn clause [Selman and Kautz, 1991b].

3 General Framework

Computing Horn approximations is just one kind of knowledge compilation. This section defines a general framework for approximating a knowledge base. The next section presents a number of different instances of this general framework.

A *knowledge compilation system* is a tuple $\langle \mathcal{L}, \models, \mathcal{L}_{\mathrm{S}}, \mathcal{L}_{\mathrm{T}}, \mathcal{L}_{\mathrm{Q}}, f_{\mathrm{L}}, f_{\mathrm{U}} \rangle$ containing the following components:

$\mathcal{L}$ is a formal language. We identify a language with the set of all its sentences.

$\models$ is a consequence relation over sets of sentences in $\mathcal{L}$. In most of the examples we will study $\models$ has its usual meaning of logical entailment, but the framework allows $\models$ to represent other relationships, such as subsumption.

$\mathcal{L}_{\mathrm{S}}$ is the "source" sublanguage of $\mathcal{L}$, used to express a general knowledge base.

$\mathcal{L}_{\mathrm{T}}$ is the "target" sublanguage of $\mathcal{L}$, used to express approximations to the general knowledge base. It should be easier in some sense (analytically or empirically) to determine if a query is a consequence of a set of sentences in $\mathcal{L}_{\mathrm{T}}$ than of a set sentences in $\mathcal{L}_{\mathrm{S}}$.

$\mathcal{L}_{\mathrm{Q}}$ is the "query" sublanguage of $\mathcal{L}$.

f_{L} is a function mapping a theory to a (potentially infinite) sequence of (better and better) lower-bounds (defined analogously to the Horn case). Ideally the last element of the sequence is the greatest lower-bound (GLB).

f_{U} is a function mapping a theory to a (potentially infinite) sequence of (better and better) upper-bounds (again, defined analogously to the Horn case). Ideally the last element of the sequence is the least upper-bound (LUB).

Suppose a knowledge compilation system is presented a query $\alpha \in \mathcal{L}_{\mathrm{Q}}$ after performing i compilation steps on the source theory Σ. Let Σ_{lb} be the *i-th* element of $f_{\mathrm{L}}(\Sigma)$ (or the last element if i is greater than the length of $f_{\mathrm{L}}(\Sigma)$), and Σ_{ub} be the *i-th* (or similarly the last) element of $f_{\mathrm{U}}(\Sigma)$. As described earlier, if $\Sigma_{\mathrm{lb}} \not\models \alpha$ then the system answers "no", if $\Sigma_{\mathrm{ub}} \models \alpha$ then the system answers "yes", and otherwise it answers "unknown". A definite answer of "yes" or "no" always agrees with the answer to the question "does $\Sigma \models \alpha$?", even if the lower (upper) bound is not the greatest (least) bound.

The case of knowledge compilation using propositional Horn clauses fits into this framework as follows: $\mathcal{L}$ is propositional logic, $\models$ is propositional entailment, $\mathcal{L}_{\mathrm{S}}$ is the set of clauses, $\mathcal{L}_{\mathrm{T}}$ is the set of Horn clauses, and $\mathcal{L}_{\mathrm{Q}}$ contains two kinds of formulas: (1) conjunctions of clauses (CNF); and (2) disjunctions of conjunctions of literals (DNF), where each disjunct contains at most one negative literal. (In this case the query language is strictly more expressive than either $\mathcal{L}_{\mathrm{T}}$ or $\mathcal{L}_{\mathrm{S}}$. The negation of such queries, however, is equivalent to a set of Horn clauses of the same size as the original query, so the linear time algorithm for satisfiability of Horn clauses can be used for query answering.) f_{L} is given by the sequence of candidate Horn-strengthenings of Σ, as in the previous section, where the last element of the sequence is in fact the GLB. Similarly, f_{L} is the sequence of larger and larger sets of Horn resolvants where the last such set is the LUB.

4 Other Instances of Knowledge Compilation

4.1 Restricted Clausal Forms

An entire class of knowledge compilation systems can be realized by generalizing the algorithms for the propositional Horn case. The idea of a Horn-strengthening is generalized as follows:

Definition: θ-strengthening
Let θ be a particular set of clauses. A clause in that set is called a θ-clause. A clause C_{T} is a θ-strengthening of a clause C iff $C_{\mathrm{T}} \subseteq C$ and there is no θ-clause C'_{T} such that $C_{\mathrm{T}} \subset C'_{\mathrm{T}} \subseteq C$. A θ-strengthening of a set of clauses is a set of θ-strengthenings of each clause.

A knowledge compilation system is created by letting the source language $\mathcal{L}_{\mathrm{S}}$ be the set of arbitrary clauses, and the target language $\mathcal{L}_{\mathrm{T}}$ be such a clause set θ. Suppose such a θ is closed under resolution (*i.e.*, the resolvents of two θ-clauses is a θ-clause). Then the completeness theorem for resolution [Lee, 1967] tells us that *any* (non-tautologous) clause C entailed by a set of θ-clauses Σ must be subsumed by a clause C' which has a resolution proof from Σ; and therefore C' is a θ-clause itself. In other words, Σ entails a θ-strengthening of C. Further suppose that any clause in the source language has a θ-strengthening. Then the GLB in terms of θ-clauses of a theory Σ can be computed by searching for a (local) maximum in the space of θ-strengthenings of Σ. This proves the following theorem:

Theorem 1 *Let Σ be any set of clauses and θ a set of clauses such that (i) the resolvant of any two clauses in θ is also in θ, and (ii) any clause in Σ is subsumed by some clause in θ. Then Σ_{glb} is a greatest lower-bound in terms of θ-clauses iff Σ_{glb} is equivalent to a θ-strengthening of Σ, and there is no θ-strengthening Σ'_{glb} such that $\Sigma_{\mathrm{glb}} \models \Sigma'_{\mathrm{glb}}$ and $\Sigma'_{\mathrm{glb}} \not\models \Sigma_{\mathrm{glb}}$.*

Horn, reverse Horn (clauses containing at most one *negative* literal), and clauses containing two or fewer literals are examples of tractable classes of propositional clauses that meet the conditions of this theorem.

Recall that the Horn LUB of a theory is equivalent to the set of all its Horn resolvants. This is due to the resolution completeness theorem and the fact that propositional Horn clauses are closed under *subsumption*. The latter condition is necessary because one could otherwise imagine cases where resolution only yielded non-Horn clauses which subsumed the entailed Horn clauses. Thus the generalized computation of the Horn LUB is based on the following theorem:

Theorem 2 *Let Σ be any set of clauses and θ a set of clauses such that if $C \in \theta$ and C' subsumes C, then $C' \in \theta$. Then Σ_{lub} is the least upper-bound in terms of θ-clauses of Σ iff Σ_{lub} is equivalent to the set of θ resolvants of Σ.*

As we noted before, it is not actually necessary to compute *all* the θ resolvants of Σ, but only a subset logically equivalent to the entire set, as in the Horn case. All of the classes of restricted clauses mentioned above meet the condition of being closed under subsumption.

Another interesting class which satisfies theorems 1 and 2 consists of clauses *not* containing a given set of propositional letters. (The θ-strengthening of the clause p, where p is a prohibited letter, is the empty clause — that is, false.) While such a class may not have better worst-case complexity than the unrestricted language, it may be empirically desirable to "compile away" certain propositions. Subramanian and Genesereth [Subramanian and Genesereth, 1987] present a formal system for inferring that certain propositions are *irrelevant* to the computation of a given class of queries. Given this sense of what is irrelevant, knowledge compilation can then be used as way to remove the irrelevant information and simplify the theory.

The class of clauses not containing a given set of letters has the following special property, which follows from Craig's Interpolation Lemma [Craig, 1955]: The LUB is just as good as the original theory for answering queries not containing the prohibited letters. Formally:

Theorem 3 *Let Σ be a set of clauses and both θ and $\mathcal{L}_{\text{Q}}$ sets of clauses not containing a given set of propositional letters. Suppose $\alpha \in \mathcal{L}_{\text{Q}}$, and Σ_{lub} is the the least upper-bound in terms of θ-clauses of Σ. Then $\Sigma \models \alpha$ iff $\Sigma_{\text{lub}} \models \alpha$.*

Example: Let the set of propositions be instantiations of the two-place predicates F and R over a given set of individuals $\{a, b, c\}$. We interpret F as meaning "father of", and R as meaning "related to". The following axiom schema capture the usual relationship between these predicates, where the variables x, y, and z are instantiated over the given set of individuals:

$$\begin{array}{l} F(x,y) \supset R(x,y) \\ R(x,y) \wedge R(y,z) \supset R(x,z) \\ R(x,y) \supset R(y,x) \end{array}$$

Suppose Σ consists of these axioms together with a set of facts about specific father-of relationships: $\{F(a,b), F(b,c) \vee F(a,c)\}$. If it were known that queries would concern only the related-to predicate, it would be useful to compute the LUB that prohibits the letter F. This yields a theory (equivalent to) the last two axiom schema (transitivity and reflexivity) together with the the facts $\{R(a,b), R(b,c)\}$.

4.2 First-Order Theories

A further natural extension of restricted clausal forms is to first-order languages. In the first-order case, the worst-case complexity of the general and restricted languages are the same. For example, satisfiability for first-order clauses and first-order Horn clauses is equally undecidable.[4]

[4]A *first-order clause* is a sentence in prenex form containing only universal quantifiers, whose matrix is a disjunction of (first-order) literals. A *first-order Horn clause* is a first-order clause containing at most one positive literal.

None the less, the restricted language may have better average-case complexity for a particular domain. In this section we will examine the issues that arise in a straightforward extension of restricted clausal knowledge compilation systems to the first-order case.

We have seen how the greatest-lower bound of a theory Σ can be computed by searching through the space of $\mathcal{L}_T$-strengthenings of Σ. In order to decide if a candidate strengthening $\Sigma'_{\rm lb}$ is a better lower-bound than the current strengthening $\Sigma_{\rm lb}$, the algorithm must check if $\Sigma_{\rm lb} \models \Sigma'_{\rm lb}$. In the first-order case this test is not recursive, so the simple search procedure could become "stuck" at $\Sigma_{\rm lb}$, even if it is not the greatest lower-bound.

Fortunately, it is not difficult to avoid this problem. The computation steps to determine $\Sigma_{\rm lb} \models \Sigma'_{\rm lb}$ should be interleaved with the generation and test of the lexicographically next candidate $\mathcal{L}_T$-strengthening $\Sigma''_{\rm lb}$. If the test $\Sigma_{\rm lb} \models \Sigma''_{\rm lb}$ returns "true", the computation for $\Sigma_{\rm lb} \models \Sigma'_{\rm lb}$ can be abandoned. Similarly the computation for $\Sigma_{\rm lb} \models \Sigma''_{\rm lb}$ can be interleaved with the generation and test of the following candidate strengthening, and so forth. With this modification the generalized **GLB algorithm** is correct for any knowledge compilation system where $\mathcal{L}_S$ consists of first-order clauses, and $\mathcal{L}_T$ is a class of first-order clauses which satisfies the conditions of Theorem 1.

The basic **LUB algorithm** need not be modified for first-order target languages that are closed under subsumption. Theorems 2 and 3 thus hold true for first-order as well as for propositional clauses, where "propositional letter" is understood to mean "predicate". While the classes based on restricting the length of the clauses (or the predicates contained in the clauses) are close, the first-order Horn and reverse Horn classes are *not*. For example, the non-Horn clause $\forall x, y \,.\, P(x,b) \vee P(a,y)$ properly subsumes the Horn clause $P(a,b)$. Therefore the LUB in the first-order Horn case must also incorporate Horn clauses that are subsumed by non-Horn resolvants, as stated in the following theorem.

Theorem 4 *A set of first-order clauses $\Sigma_{\rm lub}$ is the least Horn upper-bound of a set of first-order clauses Σ iff $\Sigma_{\rm lub}$ is equivalent to the set of Horn clauses subsumed by resolvants of Σ.*

Similar conditions apply to the reverse Horn case. In [Selman and Kautz, 1991b] we present an algorithm for generating the LUB based on theorem 4.

Finally, we note that the query language $\mathcal{L}_Q$ should be such that the negation of a query falls in $\mathcal{L}_T$. Thus, in general, the queries will be existentially-quantified sentences whose matrix is a subset of DNF. For example, in the Horn case, each disjunct contains at most one negative literal; in the binary clause case, each disjunct contains at most two literals; and so on.

4.3 Definite Clauses and Logic Programs

A *definite clause* is a clause containing exactly one positive literal. Languages based on definite clauses can be efficiently implemented and have found widespread practical applications. For example, function-free first-order definite clauses form the basis of the database language "datalog" [Ullman, 88], and general first-order definite clauses form the basis of the programming language Prolog. (In both cases, non-logical operators such as "cut" and "failure to prove" extend the logical basis.)

It is not possible, in general, to find a definite clause lower-bound of a general theory. For example, there is no definite clause lower-bound for the theory $\{\neg p\}$.

Least upper-bounds do exist, as they do for *any* source and target languages — because the empty set is a trivial upper-bound, and the least upper-bound is equivalent to the union of all upper-bounds. However, definite clauses are not closed under subsumption; for example, the non-definite clause $\neg p$ subsumes the definite clause $\neg p \vee q$. Therefore the original **LUB algorithm** is not sufficient. A version of theorem 4 *does* hold, where "Horn" is replaced by "definite clause", and can be used as the basis for an algorithm (for both the propositional and first-order cases). For example, the definite clause LUB of $\{\neg p\}$ is the set of all (binary) definite clauses subsumed by $\neg p$, *i.e.,* $\{\neg p \vee q,\ \neg p \vee r,\ \neg p \vee s,\ \ldots\}$. In effect, $\neg p$ is approximated by "p implies anything," since the LUB is equivalent to $\{p \supset q,\ p \supset r,\ p \supset s,\ \ldots\}$.

Both lower and upper bounds do exist when the source language as well as the target language consists of definite clauses. Some recent research on the analysis of Prolog programs (for use in, for example, optimization and program specification) can be viewed as a kind of knowledge compilation. For example, [Heintze and Jaffar, 1990] describes how to construct a *recursive* (*i.e.,* decidable) approximation to a potentially non-recursive logic program. Their method is based on modifying each predicate by relaxing the relationship between the arguments to the predicate. For example, if a logic program computes (entails) $\{P(a,b) \wedge P(c,d)\}$, the approximation computes (entails) $\{P(a,b) \wedge P(c,d) \wedge P(a,d) \wedge P(c,b)\}$. Thus their method computes a lower-bound of the theory (but not in general the greatest lower-bound).

4.4 Terminological Reasoning

We now consider frame-based knowledge representation languages as studied in [Levesque and Brachman, 1985] (see also [Patel-Schneider *et al.*, 1990]).

Levesque and Brachman consider a language $\mathcal{FL}$ in which one can describe structured concepts in terms of other concepts, either complex or primitive. For example, if we wished to describe people whose male friends are all doctors with some specialty, we could use the concept:

```
(person with every male friend is a (doctor with a specialty)).
```

This concept is captured in $\mathcal{FL}$ by

```
(AND person
     (ALL (RESTR friend male)
          (AND doctor
               (SOME specialty)))),
```

which contains all the constructs (the capitalized terms) used in the language. Levesque and Brachman consider the complexity of determining whether one concept *subsumes* another. For example, the concept

```
(person with every male friend is a doctor)
```

subsumes the one given above. Note that this can be determined without really knowing anything about the various concepts used in these descriptions. Now, their central technical result is that determining subsumption in $\mathcal{FL}$ is intractable, but that removing the `RESTR` construct leads to polynomial time computable subsumption. The restricted language is called $\mathcal{FL}^-$.

So for efficient subsumption, one can use the language $\mathcal{FL}^-$. But this language may not be sufficiently expressive for practical applications. Knowledge compilation provides again an alternative. In this case the idea is to take a concept description in the language $\mathcal{FL}$ and to approximate it using two concept descriptions in $\mathcal{FL}^-$: a best lower-bound, *i.e.*, the most general more specific concept in $\mathcal{FL}^-$, and a best upper-bound, *i.e*, the most specific more general (subsuming) concept in $\mathcal{FL}^-$.

As an example consider the first concept given above. It is not difficult to see that the concepts `person` and

```
(AND person
     (ALL friend
          (AND doctor
               (SOME specialty))))
```

in $\mathcal{FL}^-$ are examples of, respectively, an upper-bound and a lower-bound in $\mathcal{FL}^-$. (These are also the best bounds in this case.) The system can store such bounds with the original concept description, and use them to try to determine quickly whether the newly given concept subsumes it or is subsumed by it.

More formally, we are dealing with the knowledge compilation system $\langle \mathcal{FL}, \Rightarrow, \mathcal{FL}, \mathcal{FL}^-, \mathcal{FL}, f_{\mathrm{L}}, f_{\mathrm{U}} \rangle$, in which $\Rightarrow$ stands for "is subsumed by." We are currently working on anytime compilation algorithms for computing the functions f_{L} and f_{U}. (Note that queries are assumed to be in $\mathcal{FL}$. We conjecture that the subsumption relation between a concept given in $\mathcal{FL}$ and one in $\mathcal{FL}^-$ (*i.e.*, one of bounds) can be determined efficiently.)

So far, we have treated our knowledge base as containing only a single concept description. In general, a KB will of course contain a hierarchy of concepts [Brachman *et al.*, 1990]. In that case, we simply store bounds with each concept. When given a concept, the system can use those bounds in determining the appropriate place of the new concept in the hierarchy.

5 Conclusions

We introduced the notion of knowledge compilation. The central idea behind knowledge compilation is to translate (compile) declarative knowledge into a more efficient form. A unique advantage of our approach is the use of both lower and upper bounds can speed reasoning without giving up correctness or completeness.

We discussed various concrete examples of our approach. In particular, we showed how the procedures for compiling propositional theories into Horn theories [Selman and Kautz, 1991a] can be generalized to apply to other tractable classes of clauses. Those classes were characterized using various closure conditions. The classes containing reverse-Horn clauses, clauses with two or fewer literals, or clauses not containing a certain set of "irrelevant letters" are examples of classes that satisfy the closure conditions. We also showed how to modify our approach to handle the first-order case. Finally, we discussed the compilation of concept descriptions given in a terminological representation languages. This example showed that our knowledge compilation approach appears suited for dealing with a large variety of knowledge representations language — not only traditional logics.

Currently we are working on empirical evaluation of our knowledge compilation approach. The domain under study is a small part of Forbus's qualitative process theory [Forbus, 1984]. Qualitative axiomatizations of the physical world can be used to reason about everyday activities such as boiling water, rolling balls, or perhaps even the steeplechasing crocodile mentioned in the introduction. We hope to automatically convert the original non-Horn axiomization of qualitative process theory into Horn approximations; in essence, automating part of the work that Forbus performed when implementing his theory using the ATMS [Forbus, 1990].

References

[Boddy and Dean, 1988] Mark Boddy and Thomas Dean. Solving time dependent planning problems. Technical report, Department of Computer Science, Brown University, 1988.

[Brachman *et al.*, 1990] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with classic: When and how to use a kl-one-like language. In J. Sowa, editor, *Formal Aspects of Semantic Networks*. Morgan Kaufmann, 1990.

[Cook, 1971] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[Craig, 1955] W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22, 1955.

[Dowling and Gallier, 1984] William F. Dowling and Jean H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formula. *Journal of Logic Programming*, 3:267–284, 1984.

[Doyle and Patil, 1991] J. Doyle and R. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–298, 1991.

[Forbus, 1984] K.D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[Forbus, 1990] Kenneth D. Forbus. The qualitative process engine. In Deaniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 220–235. Morgan Kaufmann, Los Altos, CA, 1990.

[Heintze and Jaffar, 1990] Nevin Heintze and Joxan Jaffar. A finite presentation theorem for approximating logic programs. In *Proceedings of POPL-90*, page 197, 1990.

[Lee, 1967] R. C. T. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable From Given Axioms*. PhD thesis, University of California at Berkeley, Berkeley, CA, 1967.

[Levesque and Brachman, 1985] H.J. Levesque and R.J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In R.J. Brachman and H.J. Levesque, editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufmann, Los Altos, CA, 1985.

[Levesque, 1989] Hector J. Levesque. Logic and the complexity of reasoning. Technical Report KRR-TR-89-2, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, Jan 1989.

[Patel-Schneider *et al.*, 1990] P. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. MacGregor, W.S. Mark, D.L. McGuinness, B. Nebel, A. Schmiedel, and J. Yen. Term subsumption languages in knowledge representation. *AI Magazine*, 11(2):16–23, 1990.

[Selman and Kautz, 1991a] Bart Selman and Henry Kautz. Knowledge compilation using horn approximations. In *Proceedings of AAAI-91*, Anaheim, CA, 1991.

[Selman and Kautz, 1991b] Bart Selman and Henry Kautz. Methods of knowledge compilation. In Preparation, 1991.

[Subramanian and Genesereth, 1987] Devika Subramanian and Michael R. Genesereth. The relevance of irrelevance. In *Proceedings of IJCAI-87*, volume 1, page 416, 1987.

[Ullman, 88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, Rockville, MD, 88.