The Hyper System: Knowledge Reformation for Efficient First-order Hypothetical Reasoning

Helmut Prendinger Mitsuru Ishizuka Tetsu Yamamoto

Department of Information and Communication Engineering
School of Engineering, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: {helmut,ishizuka,tetsu}@miv.t.u-tokyo.ac.jp

Abstract. We present the HYPER system that implements a new approach to knowledge compilation, where function-free first-order Horn theories are transformed to propositional logic. The compilation method integrates techniques from deductive databases (relevance reasoning) and theory transformation via unfold/fold transformations, to obtain a compact propositional representation. The transformed theory is more compact than the ground version of the original theory in terms of significantly less and mostly shorter clauses. This form of compilation, called *knowledge (base) reformation*, is important since the most efficient reasoning methods are defined for propositional theories, while knowledge is most naturally expressed in a first-order language. In particular, we will show that knowledge reformation allows low-order polynomial time inference to find a near-optimal solution in cost-based first-order hypothetical reasoning (or 'abduction') problems. We will also present experimental results that confirm the effectiveness of our compilation method.

Keywords. knowledge representation, knowledge compilation, knowledge-based systems, abduction

The Hyper System: Knowledge Reformation for Efficient First-order Hypothetical Reasoning

Abstract. We present the HYPER system that implements a new approach to knowledge compilation, where function-free first-order Horn theories are transformed to propositional logic. The compilation method integrates techniques from deductive databases (relevance reasoning) and theory transformation via unfold/fold transformations, to obtain a compact propositional representation. The transformed theory is more compact than the ground version of the original theory in terms of significantly less and mostly shorter clauses. This form of compilation, called *knowledge (base) reformation*, is important since the most efficient reasoning methods are defined for propositional theories, while knowledge is most naturally expressed in a first-order language. In particular, we will show that knowledge reformation allows low-order polynomial time inference to find a near-optimal solution in cost-based first-order hypothetical reasoning (or 'abduction') problems. We will also present experimental results that confirm the effectiveness of our compilation method.

1 Introduction and Motivation

The need for knowledge reformation derives from two facts about declarative representations of knowledge. First, representations are designed for a variety of queries; hence, they will contain information that is not relevant to answering some particular query or query type (Levy *et al.* [8], see also the literature on semantic query optimization [2]). Second, many interesting problems in artificial intelligence require the representational power of first-order theories, but it is well-known that reasoning with such theories is computationally expensive (Levesque [7]). On the other hand, considerable progress has been made in developing efficient mechanisms for *propositional* reasoning. For instance, GSAT is an efficient procedure for solving propositional satisfiability problems (Selman and Kautz [14]), and the NBP and SL methods are fast mechanisms for solving propositional hypothetical (or 'abductive') reasoning problems (Ishizuka and coworkers [9, 5]). Recall that hypothetical reasoning is NP-hard, even for very basic forms of propositional problems (Eiter and Gottlob [4]). The aim of knowledge reformation is to preserve the generality and compactness of *representing* knowledge in first-order Horn logic, while at the same time allow for *processing* a highly efficient propositional knowledge base (KB). Knowledge reformation extends existing work on *knowledge compilation* (Cadoli and Donini [1]) to the first-order case. Compilation methods preprocess a propositional KB off-line such the result can be used to speed up on-line query answering. By contrast, we start with a first-order KB and generate a propositional KB of manageable size.

In principle, the idea of knowledge reformation can be implemented by a transformation that instantiates variables in first-order theories by constants ('grounding'). If no non-zero function symbols are allowed, a finite set of essen-

tially propositional clauses is obtained. This simple-minded approach is obviously impractical since a huge number of propositional clauses will be produced (Levesque [7]). Therefore, we suggest to apply theory transformation (Tamaki and Sato [16]) before actually instantiating the theory. Specifically, the principled application of unfold—definition—fold transformation steps eliminates 'unnecessary' variables, i.e., variables that occur in the body B but not in the head H of a clause $H \leftarrow B$. Since the instantiation of a clause C is exponential in the number of different variables occurring in C, the clauses resulting from transforming C allow for significantly less ground clauses than instantiating C. Let the original clause (with four different variables) be

$$C_0: \quad q(X,Y) \leftarrow p1(X,Z1) \wedge p2(Z1,Z2) \wedge p3(Z2,Y)$$

that yields $\mathcal{O}(n^4)$ clauses upon instantiation. Transformation replaces C_0 by

$$C_1: \quad q(X,Y) \leftarrow newp(X,Z2) \wedge p3(Z2,Y)$$
$$C_2: \quad newp(X,Z2) \leftarrow p1(X,Z1) \wedge p2(Z1,Z2)$$

that result in $\mathcal{O}(2n^3)$ instantiated clauses since both C_1 and C_2 have three different variables each. The reformation procedure is combined with the *Slide-down and Lift-up* (SL) method, an efficient hypothetical reasoning method (Ishizuka and Matsuo [5]).

The main contribution of knowledge reformation for the KR community is twofold. First, propositional theories can be derived *automatically* from their first-order pendants. Consequently, researchers interested in propositional algorithms may approach problems traditionally formulated in function-free first-order Horn logic without extra effort. Second, the resulting propositional theories have attractive computational properties in terms of small size and shorter clauses, and can thus be processed very efficiently. Interestingly, our transformation procedures may serve a similar function to *knowledge object decomposition* described by Debenham [3], who uses decomposition for KB maintenance. Our reformation procedure can also be used as a translator from first-order to propositional form, which is planned for the TPTP Problem Library (Sutcliffe and Suttner [15]).

The paper is organized as follows. Section 2 describes the configuration of the HYPER system. In Section 3, we show how techniques from relevance reasoning can be used to rule out parts of a KB that are not related to a set of queries. Section 4 introduces the transformation procedures by example. In Section 5, we describe how clauses are actually instantiated, as a by-product of constructing the so-called query-tree for a query type. Section 5 reports on our experimental results. In Section 6, we briefly discuss and summarize the paper.

2 System Configuration

Computational efforts in the HYPER system are divided into an *off-line* and an *on-line* phase, as illustrated in Fig. 1. HYPER is an acronym for "Hypothetical

Off-line phase.
Input: first-order hypothetical reasoning problem.

- *Relevance reasoning.* Isolate part of KB relevant to query type $p(\bar{X})$.
- *Theory transformation.* Apply unfold/fold transformation steps.
- *Instantiation.* Generate ground theory by constructing query-tree.

On-line phase.
Input: propositional hypothetical reasoning problem.

- *Relevance reasoning.* Isolate part of KB$'$ relevant to query $p(\bar{c}_i)$.
- *Hypothetical reasoning.* Apply the SL method to generate a near-optimal solution to the hypothetical reasoning problem.

Fig. 1. Configuration of the HYPER system.

reasoning employing reformation". In the off-line (preprocessing) phase we first isolate the portion of the first-order KB that is relevant to answering some query *type* [8, 13]. A query type $p(\bar{X})$ is like a ground query such as $p(a, b)$, but with all constants replaced by variables. Next, the KB is transformed via unfold/fold transformation steps [16, 12]. Finally, the transformed theory is instantiated as a by-product of constructing the query-tree [8]. The output of the off-line phase is the ground (i.e., propositional) knowledge base KB$'$ relevant to all instantiations $p(\bar{c_1}), ..., p(\bar{c_n})$ of the query type that may have a solution with respect to the hypothetical reasoning problem.

The on-line phase assumes a ground query such as $p(\bar{c_i})$ and computes all propositional clauses relevant to the ground query. Then, the SL method is applied to actually generate a solution to some (cost-based) hypothetical reasoning problem.

3 Relevance Reasoning

In this section we will introduce procedures that (i) partition a Horn theory into (independent) subtheories, and (ii) remove clauses from a subtheory that cannot contribute to the solution of any query, also called *strongly (proof-based) irrelevant* clauses (Levy *et al.* [8], Schurz [13]).

We consider first-order Horn theories T, i.e., sets of clauses C of the form "$q(\bar{X}_{n+1}) \leftarrow p_1(\bar{X}_1) \wedge ... \wedge p_n(\bar{X}_n)$" where $q(\bar{X}_{n+1}), p_1(\bar{X}_1), ..., p_n(\bar{X}_n)$ are atomic formulas, and $\bar{X}_i$ denotes the sequence of variables $X_{i,1}, ..., X_{i,m_i}$. The atom $q(\bar{X}_{n+1})$ is called the *head* of the clause, denoted by $hd(C)$, the conjunction $p_1(\bar{X}_1) \wedge ... \wedge p_n(\bar{X}_n)$ is called the *body* of the clause, denoted by $bd(C)$. The variables occurring in a clause are implicitly universally quantified. A clause (theory) containing no variables is called *ground* (or simply propositional). A clause C is called function-free if it does not contain non-zero function symbols (i.e., constants are allowed). Let $\mathcal{V}(hd(C))$ denote the set of variables occurring in the head of a clause C, and $\mathcal{V}(bd(C))$ the set of variables occurring in the

clause body. A clause C is range-restricted if $\mathcal{V}(hd(C)) \subseteq \mathcal{V}(bd(C))$. All clauses considered here are Horn, function-free, and range-restricted. Moreover, we impose the restriction that theories are *acyclic*, i.e., the corresponding directed graph contains no cycles.

In the first phase of the reformation process, the first-order theory T is factorized as follows: (i) if T is tree-structured, it can be partitioned into independent subtheories; (ii) if T is acyclic, a subtheory is generated for each query type $p(\bar{X})$. A tree-structured theory T is split into disjoint subtheories $T_1, ..., T_n$ such that no clause C in a given subtheory T_i resolves with some clause D from a different subtheory T_j. This means that the search space for a given atomic query type $p(\bar{X})$ can be restricted to a single subtheory T_i. In the more general case of acyclic theories, factorizing is performed by means of an algorithm that computes all clauses that are 'reachable' from a query type. Informally, a clause C is *reachable* from a query type $p(\bar{X})$ if there exists some path from $p(\bar{X})$ to the head of C. Observe that theory factorizing can be done in polynomial time.

This concludes the first reformation phase. The procedures introduced so far already significantly reduce the number of clauses that have to be considered when answering a ground query. It is important to note that for acyclic T, factorizing can be done by only considering the query types $p(\bar{X})$, and with tree-structured T, even independent of a query type.

4 Theory Transformation via Variable Elimination

The motivation for applying unfold/fold transformations [16, 12] is to reduce the complexity of a theory *as measured by the number of possible ground instantiations* of clauses. Since a clause C is exponential in the number n of different variables occurring in C, we try to minimize n. Theory transformation is an equivalence-preserving form of transformation [16].

Theory transformation proceeds by successively applying unfold (optional), definition, and fold rules, in that order, to the theory. More specifically, the application of transformation rules eliminates *unnecessary* variables, i.e., variables that occur in the body $bd(C)$ but not in the head $hd(C)$ of a clause C. One algorithm described in Proietti and Pettorossi [12] eliminates all unnecessary variables, but the algorithm is not guaranteed to halt. We will suggest terminating procedures that eliminate a sufficient number of unnecessary variables. In particular, we introduce a novel set of procedures that automatize the definition rule for a broad class of clause bodies. There is a lot of technical detail involved in describing the procedures (see Prendinger and Ishizuka [10]). For brevity, we only show the result of performing definition and folding steps. For definitions of unfolding, definition and folding rules, see Tamaki and Sato [16].

We start with some terminology to distinguish different kinds of clause bodies. The distinction is intended to cover a broad range of possible clause bodies. The most central notion is that of a *block* of a clause body [12].

Block. Given a clause C and a set B of atoms in $bd(C)$. We define a binary relation R over B such that: given two atoms B_1 and B_2 in B, $R(B_1, B_2)$ if

and only if $\mathcal{V}(B_1) \cap \mathcal{V}(B_2) \neq \emptyset$. We let R^* denote the reflexive and transitive closure of R over $bd(C)$. By $partbd(C)$ we denote the partition of the body of C into blocks w.r.t. R^*. Note that each variable occurs in at most one block of $partbd(C)$. For instance, let C be the clause

$$q(X, Y, Z) \leftarrow p1(X, X1) \wedge p2(Y, Y1) \wedge p3(X1, Z)$$

Here $partbd(C)$ has two blocks, $\{p1(X, X1), p3(X1, Z)\}$ and $\{p2(Y, Y1)\}$. Consider a clause

$$C_0 : \ q(X, Y) \leftarrow p1(X, Z1, Z2) \wedge p2(Z1) \wedge p3(Z2, Y)$$

where the clause body forms a single block. The new clause

$$C_1 : \ newp(X, Z2) \leftarrow p1(X, Z1, Z2) \wedge p2(Z1)$$

is generated by the definition rule ($newp/2$ a fresh predicate symbol), and then folded with C_0. By folding C_1 with C_0, we obtain

$$C_2 : \ q(X, Y) \leftarrow newp(X, Z2) \wedge p3(Z2, Y)$$

The transformed theory consists of the two clauses C_1 and C_2, that allow for significantly less instantiations than the originial theory (C_0).

A block can have a variety of syntactical forms.

Chain. An example for a chain was given in the introductory section. We will revisit a chain example in the section on experiments.

Isolated blockpart. Consider the clause

$$C_0 : \ q(X, Y) \leftarrow p1(X, Z) \wedge p2(Z, Y) \wedge p3(Z, Z1)$$

The variable $Z1$ in $bd(C)$ is called 'isolated' (occurs only once in C_0), and $p3(Z, Z1)$ is thus an isolated blockpart. The definition rule for isolated blochparts generates the clause

$$C_1 : \ newp(Z) \leftarrow p3(Z, Z1)$$

which is folded with C_0 to produce

$$C_2 : \ q(X, Y) \leftarrow p1(X, Z) \wedge p2(Z, Y) \wedge newp(Z)$$

The theory transformation procedure is described in Prendinger and Ishizuka [10] (see also Proietti and Pettorossi [12]). The transformation procedure performs unfolding—definition—folding cycles until none of the procedures is applicable. The procedure significantly extends the algorithm in [12]. Except for a special instance of clause bodies with isolated blockparts, their algorithm did not eliminate any unnecessary variables. Observe that after applying theory transformation, we obtain slightly more first-order clauses, but they yield significantly fewer instantiations.

5 Theory Instantiation

In the last phase of the reformation process, the theory is instantiated. By employing the query-tree method (Levy *et al.* [8]), we obtain exactly the set of ground clauses relevant to a query type. A *query-tree* (QT) is a compact representation of a search tree for first-order Horn theories in the form of an AND-OR tree with goal-nodes and rule-nodes. Since we do not allow recursion in clauses, our construction of the QT is simpler than the original one in [8]. On the other hand, we allow that some leaves of the query-tree are uninstantiated. Those are typically *hypotheses* that may be assumed in order to prove a query.

The query-tree algorithm consists of two successive phases. In the *bottom-up* phase, the instantiations of the base atoms (denoting facts and hypotheses) are propagated upwards to the query type. Instantiations are conceived as constraints on the arguments of predicates. Then, in the *top-down* phase, the constraint generated for the query type is 'pushed down' along the branches of the QT. Thereby, the constraints of nodes (of predicates) in the QT might be further constrained, which possibly leads to pruning parts of the tree constructed bottom-up. In effect, we obtain exactly the set of ground clauses relevant to a query type together with all instantiations of the query type that have a solution with respect to the theory if certain hypotheses can be consistently assumed (Prendinger and Ishizuka [11]). The complexity of building the QT is linear in the number of rules and possibly exponential in the arity of predicates.

6 Empirical Evaluation

The impact of the savings gained by knowledge reformation is tested by means of the *Slide-down and Lift-up* (SL) mechanism (Ishizuka and Matsuo [5]), an efficient propositional hypothetical reasoning method for computing a near-optimal solution close to the optimal solution (e.g., diagnosis). In short, the SL method uses a linear programming technique (the simplex method) to determine an initial search point, and a non-linear programming technique to find a near-optimal 0-1 solution. The inference speed of the SL method is low-order polynomial, approximately $\mathcal{O}(n^{1.85})$, where n is the number of propositional variables in the problem formulation. The SL method runs on a SGI ONYX workstation (200 MHz CPU $\times$ 2, 512 MB memory). The theory factorizing, variable elimination, and instantiation procedures are implemented in C and run on the same machine.

In hypothetical reasoning (or 'abduction'), we are given a knowledge base T, hypotheses (or 'assumables') $\mathcal{H}$, and a query q. Sometimes the problem formulation contains inconsistency constraints $I \subset T$ of the form "$inc \leftarrow h_1(\bar{c}_1) \wedge ... \wedge h_n(\bar{c}_n)$" where $h_i \in \mathcal{H}$, $\bar{c}_n$ a sequence of constant symbols, and the symbol "inc" denotes the impossible state (*falsum*). The hypothetical reasoning task consists in finding minimal sets $H_1, ..., H_n$ ($H_i \subseteq \mathcal{H}$) such that (i) $T \cup H_i \vdash q$, and (ii) $T \cup H_i \nvdash inc$.

The experiments are intended to show the speedup effect due to knowledge reformation (except for factorizing which is not needed here but has been shown

elsewhere [8]). They involve simple theories that are not very interesting in themselves but allow us to make easy comparisons by varying relevant parameters, such as the number of unnecessary variables, and the number of constants in the problem formulations. We expect the efficiency gain to be the more significant the more unnecessary variables are eliminated.

As an example for a *chain*, let the theory T consist of the clauses

$$(r_1)\ \ path(X,Y) \leftarrow link1(X,Z1) \wedge link2(Z1,Z2) \wedge link3(Z2,Z3) \wedge link4(Z3,Y)$$
$$(r_2)\ \ inc \leftarrow link1(X,Y) \wedge link3(Y,X)$$

As the set of element hypotheses we take $\mathcal{H} = \bigcup_{k=1}^{4}\{link_k(a_i,a_j)/w : i,j \in \{1,...,n\}\}$. By varying the number of constants n, we obtain theories of different size. We may also look at theories with a greater (smaller) number of links in the definition of $path/2$, by varying the number k.

In our experiment, we assume $n \leq 12$ and $k = 4$ and call this instance *4-path example*. For simplicity, we assume a default value w as the cost (or weight) for all element hypotheses $h \in \mathcal{H}$, i.e., we are essentially looking for a subset-minimal solution to the hypothetical reasoning problem [4]. After applying the reformation procedure, r_1 is replaced by

$$(r_1')\ \ path(X,Y) \leftarrow newp1(X,Z3) \wedge link4(Z3,Y)$$
$$(r_1'')\ \ newp1(X,Z3) \leftarrow newp2(X,Z2) \wedge link3(Z2,Z3)$$
$$(r_1''')\ \ newp2(X,Z2) \leftarrow link1(X,Z1) \wedge link2(Z1,Z2)$$

The inconsistency constraint (r_2) and $\mathcal{H}$ remain unchanged.

The impact of knowledge base reformation on problem size and processing time for the 4-path example is summarized in Fig. 2. Here, '# consts' is the number of constants occurring in the problem formulation. '# rules' refers to the number of instantiated (propositional) rules before and after reformation. The numbers do not include the number of instantiated inconsistency constraints which is the same for both unreformed and reformed theories. '# atoms' is the number of atoms (propositional variables) in the problem formulation (including hypotheses atoms and atoms occurring in inconstistency constraints). The two columns under 'inst.-time (sec)' show the CPU times needed to instantiate the original and reformed theories, respectively, and the two columns under 'sol.-time (sec)' give the CPU times to find a near-optimal solution for an instantiated goal $path(a,b)$, for the original and reformed theories, respectively. The CPU time needed to reform the theory via the variable elimination procedures is 0.015 seconds. The relevance reasoning part of the on-line phase is integrated into the SL method.

Instantiating a clause in the original theory effectively means to create a lookup table with m^n entries, where m is the number of constants in the problem formulation, and n is the number of different variables in the clause. In the 4-path example, for instance, if there are ten constants, we obtain $10^5 = 100000$ propositional clauses corresponding to the first-order definition of a path (r_1),

# consts	# rules		# atoms		inst.-time (sec)		sol.-time (sec)	
	unref.	ref.	unref.	ref.	unref.	ref.	unref.	ref.
3	243	81	1278	306	0.24	0.18	0.75	0.71
4	1024	192	5232	688	0.57	0.33	1.04	0.70
5	3125	375	15800	1300	2.30	0.50	2.60	0.80
6	7776	648	39132	2196	9.23	0.78	7.41	0.89
7	16807	1029	84378	3430	34.08	1.02	26.62	1.04
8	32768	1536	164288	5056	119.83	1.41	98.67	1.28
9	59049	2187	295812	7128	393.22	1.91	148.48	1.62
10	100000	3000	500700	9700	1327.55	2.70	–	2.08
11	161051	3993	806102	12826	6611.00	3.58	–	2.69
12	248832	5184	1245168	16560	–	4.81	–	3.45

Fig. 2. Results for the 4-*path* example.

that has five different variables. The space-reducing effect of knowledge reformation can be seen as decreasing the exponent n. In the given example, the original clause r_1 is transformed to three clauses with three different variables in each clause, which allow for $3 \times 10^3 = 3000$ propositional clauses. Another property of the reformed part of the output theory is that the transformed clauses are *shorter*, which has a direct effect on the number of atoms in the problem formulation. The total number of atoms in the respective theories is given in Fig. 2.

Regarding the time needed to instantiate the first-order theories (which is done off-line), note that the query-tree method produces a ground theory that can be used for *all* instantiations of a particular query type that may have a solution. So, if all possible ground queries are posed, the actual time needed for each query is $\frac{t}{m^k}$, where t is the instantiation time, m is the number of constants in the problem formulation, and k is the arity of the query predicate. In the case of ten constants, for instance, the actual cost for the reformed theory is $\frac{2.7}{100} = 0.027$ seconds for each individual query (if all ground queries are posed). Although it is unrealistic to assume that all possible queries are posed, we nevertheless expect that compiled theories are *reused* a sufficiently large number of times to amortize the cost to instantiate the theory. The last two columns in Fig. 2 summarize the efficiency gain resulting from knowledge reformation in CPU seconds. The total speedup is up to a factor of 153 for nine constants.

To support our hypothesis that performance is crucially dependent on how many unnecessary variables are eliminated, we performed another experiment that is analogous to the 4-path example, but has three body atoms in the definition of $path/2$ instead of four. For this example (where less variables are eliminated) we measured speedups up to a factor of 14 (for twelve constants). Figures 3 and 4 summarize the speedup effect of the transformation procedures for blocks and isolated blockparts (using the examples of Section 4). Inference time here refers to the total time needed to solve a problem, i.e., off-line and on-line reasoning for a specific goal. For less than six constants, the speedup

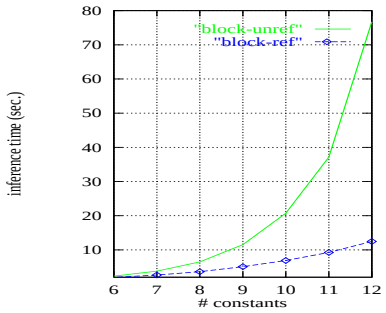

Fig. 3. Block example.

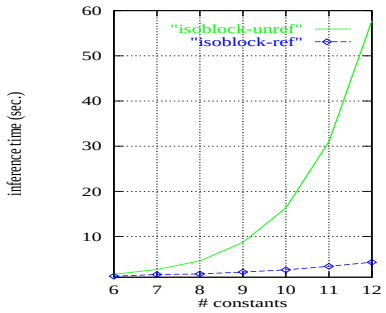

Fig. 4. Isolated blockpart example.

effect was negligible. For realistic theories, we expect (i) that our reformation procedures can be applied to a significantly large portion of the KB, and (ii) that all of our three transformation procedures are used to a varying extent.

7 Conclusion

In this paper, we address the following problem: given a problem formulation in first-order Horn logic without function symbols, how can we arrive at a compact propositional representation? In particular, we have presented the HYPER system, an effective method for cost-based first-order hypothetical reasoning problems, where a KB in the language of function-free first-order Horn logic is first compiled into a propositional KB and then an efficient propositional method is applied to compute a near-optimal solution. We have shown that off-line knowledge reformation of first-order rules is very effective in reducing the number and length of generated propositional rules. Consequently, the total inference time for solving cost-based first-order hypothetical reasoning problems can be significantly reduced.

The idea to preprocess some part of the input off-line to improve on-line efficiency is employed by many others (Williams and Nayak [17], Cadoli and Donini [1]). However, those compilation methods are restricted to the propositional case, i.e., the original KB is expressed in propositional logic. By contrast, we introduce an effective way to produce a compact propositional theory from a given *first-order* theory. In this respect, our approach shares intuitions with the 'first-order planning as (propositional) satisfiability' framework of Kautz and Selman [6]. One problem, however, applies to both approaches: the presence of too many constants makes the translation (first-order to propositional) infeasible. This problem was pointed out by Levesque [7] who imagines a knowledge base with 10^5 constants.

Although the focus of the present paper is more foundational, a subset of our compilation techniques has already been shown to be practical for diagnostic tasks (Prendinger and Ishizuka [11]). In the near future, the HYPER system will be available as free software.

References

1. Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10:137–150, 1997.
2. U. S. Charkravarthy, John Grant, and Jack Minker. Foundations of semantic query optimization for deductive databases. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–273. Morgan Kaufmann Publishers, 1988.
3. John Debenham. Knowledge object decomposition. In *Proceedings 12th International FLAIRS Conference (FLAIRS-99)*, pages 203–207, 1999.
4. Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1–2):3–42, 1995.
5. Mitsuru Ishizuka and Yutaka Matsuo. SL method for computing a near-optimal solution using linear and non-linear programming in cost-based hypothetical reasoning. In *Proceedings 5th Pacific Rim Conference on Artificial Intelligence (PRICAI-98)*, pages 611–625, 1998.
6. Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings 13th National Conference on Artificial Intelligence (AAAI-96)*, 1996.
7. Hector J. Levesque. A completeness result for reasoning with incomplete first-order knowledge bases. In *Proceedings 6th Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 14–23, 1998.
8. Alon Y. Levy, Richard E. Fikes, and Yehoshua Sagiv. Speeding up inferences using relevance reasoning: a formalism and algorithms. *Artificial Intelligence*, 97:83–136, 1997.
9. Yukio Ohsawa and Mitsuru Ishizuka. Networked bubble propagation: a polynomial-time hypothetical reasoning method for computing near-optimal solutions. *Artificial Intelligence*, 91:131–154, 1997.
10. Helmut Prendinger and Mitsuru Ishizuka. Preparing a first-order knowledge base for fast inference. In *Proceedings 12th International FLAIRS Conference (FLAIRS-99)*, pages 208–121, 1999.
11. Helmut Prendinger and Mitsuru Ishizuka. Qualifying the expressivity/efficiency tradeoff: Reformation-based diagnosis. In *Proceedings 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 416–421, 1999.
12. Maurizio Proietti and Alberto Pettorossi. Unfolding—definition—folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science*, 142:89–124, 1995.
13. Gerhard Schurz. Relevance in deductive reasoning: A critical overview. In G. Schurz and M. Ursic, editors, *Beyond Classical Logic*. Academia Press, St. Augustin, 1999.
14. Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings 13th International Conference on Artificial Intelligence (IJCAI-93)*, pages 290–295, 1993.
15. G. Sutcliffe and C. B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
16. Hisao Tamaki and Taisuke Sato. Unfold/fold transformation of logic programs. In *Proceedings 2nd International Logic Programming Conference*, pages 127–138, 1984.
17. Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 971–978, 1996.