Preparing a First-order Knowledge Base for Fast Inference

Helmut Prendinger and **Mitsuru Ishizuka**
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}@miv.t.u-tokyo.ac.jp

Abstract

This paper presents an effective method to encode function-free first-order Horn theories in propositional logic. To keep the resulting theory within manageable size, we employ techniques from (ir)relevance reasoning and theory transformation. Our approach allows for the compactness of knowledge representation in first-order logic and the efficiency of propositional reasoning mechanisms. The empirical evaluation with a hypothetical reasoning mechanism indicates that our approach has the potential to solve notoriously hard problems in diagnosis, planning, and vision.

Introduction and Motivation

Declarative knowledge generated by a knowledge engineer is not always in a form that can be efficiently processed by present-day reasoning mechanisms. In particular, knowledge bases should at least have the expressiveness of first-order Horn clauses, and it is well-known that reasoning with first-order theories is computationally expensive. On the other hand, in recent years considerable progress has been made in developing highly efficient mechanisms for *propositional* reasoning. GSAT is an efficient procedure for solving propositional *satisfiability* problems (Selman & Kautz 1993); NBP and SL are fast mechanisms for solving propositional *hypothetical reasoning* problems (Santos 1994; Ohsawa & Ishizuka 1997; Ishizuka & Matsuo 1998). Hence it seems reasonable to first transform first-order theories to propositional theories and then address the task of satisfiability checking or hypotheses generation, rather than directly using the first-order theory.

Our approach is similar in spirit to recent endeavors of solving *planning* problems by propositional satisfiability algorithms (Kautz & Selman 1996). They propose various ways to encode first-order planning problems as propositional satisfiability problems. The aim of this paper is different from their work in two respects: first, we develop a *general theory* for 'encoding' first-order problems in propositional logic, as opposed to encoding particular problems such as planning problems; and second, to validate our approach we target the problem of efficient hypothetical (or 'abductive') reasoning rather satisfiability testing.

For the case of Horn theories without function symbols, a naive approach would suggest to apply all possible instantiations of constants for variables and output a (finite) set of clauses that contains no variables. These clauses can then be treated like propositional clauses. This approach is certainly infeasible since the resulting set might be prohibitively large. Therefore, our idea is to 'reform' the original first-order theory before applying all instantiations. Knowledge base *reformation* aims at reducing both (i) the number of clauses and (ii) the number of different variables in clauses. In effect, the reformed theory lends itself to significantly less instantiations and hence a smaller propositional theory. In order to reduce the number of clauses we will employ methods from *relevance reasoning* (Levy, Fikes, & Sagiv 1997; Schurz 1998), whereas procedures from *theory transformation* are used to reduce the number of variables in clauses (Tamaki & Sato 1984; Proietti & Pettorossi 1995). Relevance reasoning allows to determine the part of the theory that does not contribute to a proof of a query or a set of queries. Theory transformation eliminates 'unnecessary' variables, i.e., variables that occur in the body B but not in the head H of a clause (rule) $H \leftarrow B$.

Theory reformation is intended as an off-line process that can be used to speed up on-line computation. Therefore, theory reformation is a form of *knowledge compilation*. Most importantly, the reformed theory is generated independently of specific queries, and thus has to be done only once for certain query *types*.

The paper is organized as follows. In the next section, some terminology related to Horn logic and hypothetical reasoning is introduced. Then we show how techniques from relevance reasoning can be employed to rule out irrelevant parts of a theory. The following section describes the theory transformation procedure which eliminates unnecessary variables from clauses. By means of a small experiment we show the efficiency gain resulting from reformation. In the last section, we briefly discuss and conclude the paper.

Preliminaries

Logic

A first-order Horn clause C has 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 Horn theory T is a set of Horn clauses. Since we deal with Horn clauses (theories) only, we will sometimes omit the reference to Horn and simply speak of clauses (theories). A clause C is function-free if it does not contain function symbols. 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, non-recursive, and range-restricted. Moreover, we impose the restriction that the set of clauses is acyclic (i.e., the corresponding directed graph contains no cycles). If the head of a clause C does not occur elsewhere in T, C is called a *definition* clause. A clause (theory) containing no variables is called *ground* (or simply propositional).

Hypothetical Reasoning

In *hypothetical* (or abductive) reasoning, we are given a background knowledge base T, a hypothetical knowledge base $\mathcal{H}$, and a query or goal Q. The background theory T is a Horn theory as described above. The hypothetical knowledge base $\mathcal{H}$ contains atoms of the form $h_i(\bar{t})$ ($\bar{t}$ a sequence of terms) that might be assumed in order to prove the query; atoms in $\mathcal{H}$ are sometimes called 'abducibles'. The nonmonotonicity of hypothetical reasoning is encoded by so-called *inconsistency constraints* I. The set $I \subset T$ contains clauses of the form "$inc \leftarrow h_1(\bar{t}_1) \wedge ... \wedge h_n(\bar{t}_n)$" where $h_i \in \mathcal{H}$ and the symbol "inc" denotes the impossible state (the logical constant *falsum*). Informally, an inconsistency constraint expresses the fact that certain hypotheses cannot hold together.

The *task* of hypothetical reasoning consists in finding sets $H_1, ..., H_n$ ($H_i \subseteq \mathcal{H}$), such that the following conditions are satisfied: (i) $T \cup H_i \vdash Q$, and (ii) $T \cup H_i \nvdash inc$.

It has been shown that hypothetical reasoning can be used for a variety of *evidential* reasoning tasks, where some parts of a system are observed and other (not observable) parts are to be inferred (Poole 1998). Evidential reasoning tasks include diagnosis, perception (vision), and planning. In *diagnosis* we observe symptoms and want to determine the faulty parts of the system. In *perception* (vision) we are given a stream of sensor data and the task is to find a description (map) of the locations and shapes of the objects in the scene. *Planning* starts from a set of goals and searches for a set of actions that would bring about the goals.

Reformation by Relevance Reasoning

In this section we introduce procedures that reduce the number of clauses in a theory. The theoretical foundations for this kind of reduction are discussed in the area of *relevance reasoning* (Levy, Fikes, & Sagiv 1997; Schurz 1998). Specifically, all clauses that can never contribute to a solution of some problem, called *strongly irrelevant* clauses, are removed from the theory.

Theory Factorizing

The idea of *theory factorizing* is to split a theory T 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 Q can be restricted to a single subtheory T_i (assuming that Q resolves with some clause in T_i). All clauses in subtheories different from T_i are strongly irrelevant to proving Q.

Example 1 Let the original theory T be as follows, with query type $p(X, Y)$:

(r_1) $p(X,Y) \leftarrow q(X,Y)$.
(r_2) $p(X,Y) \leftarrow r(X,Y)$.
(r_3) $q(X,Y) \leftarrow q1(X,Z) \wedge h\_1(Z,Y)$.
(r_4) $r(X,Y) \leftarrow h\_2(X,Y)$.
(r_5) $r(X,Y) \leftarrow h\_3(X,Y)$.
(r_6) $s(X,Y) \leftarrow s1(X,Y)$.
(r_7) $t(X,Y) \leftarrow h\_4(X,Y)$.
(f_1) $s1(a,b)$.

Because we want to determine the subtheories independent of the particular instantiations that clauses will eventually take, the theory factorizing procedure is performed by considering only the predicate symbols in a clause, i.e., the goal types such as $p(\bar{X})$ or $q(\bar{X})$. In the resulting partition of T, the arguments of predicates are removed. Since clauses in the theory are indexed, the original clauses can be restored any time.

$$T_1 = \{\leftarrow p; p \leftarrow q; p \leftarrow r; q \leftarrow q1 \wedge h\_1; \\ r \leftarrow h\_2; r \leftarrow h\_3\}$$

$$T_2 = \{s \leftarrow s1; s1\}$$

$$T_3 = \{t \leftarrow h\_4\}$$

If the problem formulation contains inconsistency constraints $ic \in I$, the set I is factorized accordingly.

Essentially, the factorizing procedure does the following: (i) if a clause C does not resolve with any independent subset of the already generated partition, then $\{C\}$ is added as a new partition; (ii) if a clause C resolves with independent subsets $\mathcal{D}_1, ..., \mathcal{D}_k \in \mathcal{P}$, then those subsets and C form a new partition $\mathcal{P}'$. The procedure halts when all clauses of the original clause set T are processed.

Since the theory factorizing procedure generates a partition, it has to be applied only once, and further query types can be assigned to their respective subtheories. The theory factorizing procedure is an effective method to extract relevant information in *tree-structured* systems (Stumptner & Wotawa 1997). A

theory T is tree-structured if the graph corresponding to T consists of subtrees $S_1, ..., S_n$ such that each S_i has only one top node and there exists only one path from every node in S_i to the top node. For the more general class of acyclic systems, however, there is a more precise way to determine the relevant part of the theory. This is done by means of an algorithm that computes all clauses that are *reachable* from the query type. Informally, a clause C is reachable from a query type p if there exists some path from p to the head of C.

Theory Simplification

A given (independent) theory may (still) contain clauses that do not contribute to answering any query, since they contain subqueries that cannot be resolved with a fact or the head of some clause. Those clauses can be removed on grounds of irrelevance as well (Schurz 1998).

Theory simplification is best explained by continuing the previous example. Let T_1 be our initial theory

$$T_1 = \{\leftarrow p; p \leftarrow q; p \leftarrow r; q \leftarrow q1 \wedge h\_1; \\ r \leftarrow h\_2; r \leftarrow h\_3\}$$

Since $q1$ does not resolve with any clause, the clause $q \leftarrow q1 \wedge h\_1$ can be deleted from T_1. In general, we may also remove clauses that resolve with a deleted clause. After applying the simplification procedure, the remaining theory is

$$T_1' = \{\leftarrow p; p \leftarrow r; r \leftarrow h\_2; r \leftarrow h\_3\}$$

Observe that clauses having hypotheses $h \in \mathcal{H}$ in their body are not necessarily deleted, since hypotheses *may* contribute to a proof (if they are assumed). In the presence of inconsistency constraints, the set I can be simplified as well.

Theorem 1 *Given a theory T, clause C, and query type $q(\bar{X})$. If some C is removed from T by theory factorizing or theory simplification, then C is strongly irrelevant to prove $q(\bar{X})$ in T.*

In other words, theory factorizing and theory simplification provide *neccessary* conditions for strong irrelevance. In general, those procedures do not give sufficient conditions for strong irrelevance: after factorizing and simplifying a theory, the theory may still contain strongly irrelevant clauses. This is a consequence of ignoring particular instantiations of predicates. A more elaborate procedure may provide sufficient conditions as well (Levy, Fikes, & Sagiv 1997).

From now on, we assume that the first-order version of the factorized and simplified theory is available.

Reformation by Theory Transformation

The motivation for applying unfold/fold transformations is to reduce the complexity of a theory *as measured by the number of possible ground instantiations* of clauses. Since a theory is exponential in the number n of different variables occurring in each clause, we try to minimize n. More specifically, we try to eliminate *unnecessary* variables, i.e., variables that occur in the body $bd(C)$ but not in the head $hd(C)$ of a clause C. Then a clause is *uv-minimal* if has the smallest possible number n of unnecessary variables ($n \geq 0$).

Definition 1 A set of Horn clauses is *optimally reduced* if each of its clauses is *uv*-minimal.

As a motivating example, consider the clauses

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

$C_2 : newp(X, Z2) \leftarrow p1(X, Z1, Z2) \wedge p2(Z1)$

Clause C_1 can be folded with C_2 as the folding clause, yielding

$C_3 : q(X, Y) \leftarrow newp(X, Z2) \wedge p3(Z2, Y)$

For simplicity, assume that each of the variables X, Y, $Z1$, and $Z2$ can be instantiated to any of five constants. Then there exist $5 \times 5 \times 5 \times 5 = 625$ possibilities to instantiate C_1, and 125 possibilities for each of C_2 and C_3. Hence the original theory $T = \{C_1, C_2\}$ allows for 750 propositional clauses, whereas the reformed theory $T' = \{C_2, C_3\}$ has only 250 instantiations. The unnecessary variable $Z1$ in $bd(C_1)$ is eliminated. In general, clauses such as C_2 are not given in advance but have to be invented by the so-called *definition rule* (Tamaki & Sato 1984).

Variable Elimination Procedures

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. Below we suggest procedures to eliminate unnecessary variables from clause bodies. The most central notion is that of a *block* of a clause body.

Definition 2 (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 wrt. 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 $\{p2(Y, Y1)\}$.

A block can have a variety of syntactical forms.

Definition 3 (chain) Given a clause C and a partition $partbd(C)$. Let Bl be a block in $partbd(C)$ with k atoms where all atoms in the block can be grouped such that (i) for all adjacent A_i, A_j, $R(A_i, A_j)$, (ii) the first and the last atom in the (reordered) block contain at least one variable occurring in $hd(C)$, (iii) no other atom contains a variable occurring in $hd(C)$, and (iv) the intersecting variables in $R(A_i, A_j)$ are distinct. Then $Bl = \{\langle A_1, ..., A_k \rangle\}$ is called a *chain*.

Let C be the clause

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

The (single) block $\{p1(X,Z1), p2(Z1,Z2), p3(Z2,Y)\}$ forms a chain; X in $p(X,Z1)$ and Y in $p3(Z2,Y)$ are called *embracing* variables. A *loop* is a special form of a chain that has the form "$q(X) \leftarrow p_1(X,Y_1) \wedge ... \wedge p_n(Y_{n-1},X)$".

Definition 4 (isolated blockpart) Let C be the clause $A \leftarrow B_1 \wedge ... \wedge B_n$ where $bd(C)$ is a block. Given an atom B_i in $bd(C)$ such that for some $X \subset \mathcal{V}(B_i)$, $X \cap (\mathcal{V}(\{A, B_1, ..., B_n\})\backslash X) = \emptyset$. Then the variables occurring in X are called *isolated* variables in $bd(C)$, and B_i an *isolated blockpart* in $bd(C)$.

Let C be the clause

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

The variable $Z1$ in $bd(C)$ is isolated, and $p3(Z,Z1)$ is an isolated blockpart.

The following procedures automatize the definition rule for a broad class of clause bodies.

Procedure 1 (block) Given a clause C with a block $Bl \in partbd(C)$, and atoms B_i, B_j in Bl such that $R(B_i,B_j)$ (see Def. 2). If there are at least two unnecessary variables in Bl, generate a new clause

$$newp(Y_1,...,Y_m) \leftarrow B_i \wedge B_j$$

where $newp/2$ is a fresh predicate symbol and $Y_1,...,Y_m$ is defined as $(\mathcal{V}(B_i) \cup \mathcal{V}(B_j))\backslash(\mathcal{V}(B_i) \cap \mathcal{V}(B_j))$, else do nothing. Let C be the clause

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

where the clause body forms a single block (which is not a chain). The new clause

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

is generated. Observe that this procedure is nondeterministic.

Procedure 2 (chain) Given a clause C and a chain $\langle A_1,...,A_k\rangle$ $(k > 2)$ of a block in $bd(C)$. Generate a new clause

$$newp(X_1,...,X_n) \leftarrow A_1 \wedge ... \wedge A_{k-1}$$

with $X_1,...,X_n$ the embracing variables of A_1, A_{k-1}.

Procedure 3 (isolated blockpart) Let C be a clause with $p(X_1,...,X_n)$ $(n \geq 1)$ an atom in a block Bl in $bd(C)$ where $\bar{Y}$ is the set of isolated variables in Bl and $\bar{X}' = \{X_1,...,X_n\}\backslash\bar{Y}$. Generate a new clause

$$newp(\bar{X}') \leftarrow p(X_1,...,X_n).$$

As an example, consider the clause

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

where $Z1$ in $p3(Z,Z1)$ is isolated. According to the procedure, the new clause "$newp(Z) \leftarrow p3(Z,Z1)$" is generated.

The theory transformation procedure can be summarized as follows. For definitions of the unfolding and folding rules, the reader is referred to (Tamaki & Sato 1984).

THEORY TRANSFORMATION **Procedure**

Input theory T and definition clause C.
Output a set T' of transformed clauses.
Initially $\mathcal{D} = \{C\}$, $\mathcal{N} = \emptyset$, ($\mathcal{N}$ is set of already processed clauses), $T' = \emptyset$.

For each definition clause $D \in \mathcal{D}$ that contains unnecessary variables **do** {

1. **Unfolding step:** unfold some atom in the body of D using non-unit clauses in T and put the resulting clauses into a set U_D;
2. **Definition steps: for each** clause $E \in U_D$
   for each block $Bl \in partbd(E)$ where
   - Bl contains at least one unnecessary variable, and
   - Bl is not a faithful variant of the body of any clause in $\mathcal{D} \cup \mathcal{N}$,

   do {
   - **if** Bl is a chain, **apply** Proc. 2,
   - **if** Bl contains an isolated blockpart, **apply** Proc. 3,
   - **else apply** Proc. 1;

   and add the new definition rule to $\mathcal{D}$ };
3. **Folding steps: for each** clause E in U_D add to T' the clause resulting from E as follows:
   for every block Bl of $partbd(E)$ which is a faithful variant of a body of a clause N in $\mathcal{D} \cup \mathcal{N}$, fold Bl in E using N.

$\mathcal{D} = \mathcal{D}\backslash\{D\}$, $\mathcal{N} = \mathcal{N} \cup \{D\}$ }

As opposed to the procedure of (Proietti & Pettorossi 1995), the procedure is guaranteed to terminate.

Example and Empirical Evaluation

Since the speedup effect of relevance reasoning is already shown in (Levy, Fikes, & Sagiv 1997), we will briefly discuss the efficiency gain of theory transformation. Let the theory T_{path} consist of a following clauses.

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

The predicates *link1–4* are abducible. After applying the transformation procedure, r_0 can be 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 *ic* (r_2) remains unchanged. While (r_1) has three unnecessary variables, each of the three transformed clauses has only one unnecessary variable. Observe that unfolding is not needed in this example.

We performed a small experiment that is intended to show the speedup effect of theory transformation. In the figure, *number of constants* refers to the number

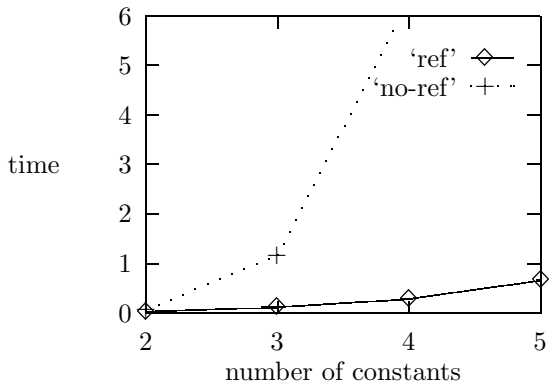

Figure 1: The effect of theory transformation in the path example. Time in seconds.

of possible bindings of *each* variable; *time* is the running time of the Networked Bubble Propagation (NBP) method (Ohsawa & Ishizuka 1997) on a Sun Ultra2 (320 MB memory). Since the number of different variables occurring in a clause is an exponential factor for the possible ground instantiations of the clause, the efficiency of reformation increases with the number of constants. It is important to note that while the number of hypotheses is constant, the number of body atoms of clauses in the reformed theory is reduced from four atoms to two atoms. Comparing theories with almost the same number of rules reveals that the theory with two body atoms per rule is more than twice as efficient as the theory with four atoms (per rule).

Discussion and Conclusion

We have presented a general theory for knowledge reformation that takes as input a first-order Horn theory and outputs a propositional Horn theory of manageable size. The reformation method is based on extensions of well-established compilation methods, originally investigated in the areas of database systems (relevance reasoning) and logic programming (theory transformation). It is important to note that theory reformation is an equivalence-preserving form of knowledge compilation that can be applied in a modular way, i.e., it is possible to apply only a subset of the available reformation procedures. Empirical results indicate that reformation may dramatically reduce the complexity of reasoning problems. The experiment deals with a rather 'pathological' case where each variable can be instantiated by any of the constants occurring in the theory. In practice, the range of possible values for attributes (ground instantiations for atoms) tends to be small. However, in (Prendinger & Ishizuka 1998) we present promising results for problem instances of model-based diagnosis.

Future research will focus both on more specialized reformation methods and particular applications. For instance, *predicate abstraction* methods allow to project out predicate arguments that are irrelevant to a set of queries, thereby reducing the number of (different) variables in a clause. On the practical side, we hope to validate our approach on various evidential reasoning tasks, in particular, on planning problems.

Acknowledgments

The first author was supported by a fellowship from the Japan Society for the Promotion of Science (JSPS).

References

Ishizuka, M., and Matsuo, Y. 1998. 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)*, 611–625.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings 13th National Conference on Artificial Intelligence (AAAI-96)*.

Levy, A. Y.; Fikes, R. E.; and Sagiv, Y. 1997. Speeding up inferences using relevance reasoning: a formalism and algorithms. *Artificial Intelligence* 97:83–136.

Ohsawa, Y., and Ishizuka, M. 1997. Networked bubble propagation: a polynomial-time hypothetical reasoning method for computing near-optimal solutions. *Artificial Intelligence* 91:131–154.

Poole, D. 1998. Learning, Bayesian probability, graphical models, and abduction. In Flach, P., and Kakas, A., eds., *Abduction and Induction: essays on their relation and integration*. Kluwer. Forthcoming.

Prendinger, H., and Ishizuka, M. 1998. Efficient diagnosis based on theory reformation and hypothetical reasoning. Submitted to the European Journal of Artificial Intelligence.

Proietti, M., and Pettorossi, A. 1995. Unfolding—definition—folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science* 142:89–124.

Santos, E. 1994. A linear constraint satisfaction approach to cost-based abduction. *Artificial Intelligence* 65:1–27.

Schurz, G. 1998. Relevance in deductive reasoning: A critical overview. *Conceptus-Studien* 13:9–56.

Selman, B., and Kautz, H. 1993. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings 13th International Conference on Artificial Intelligence (IJCAI-93)*, 290–295.

Stumptner, M., and Wotawa, F. 1997. Diagnosing tree structured systems. In *Proceedings 15th International Conference on Artificial Intelligence (IJCAI-97)*, 440–445.

Tamaki, H., and Sato, T. 1984. Unfold/fold transformation of logic programs. In *Proceedings Second International Logic Programming Conference*, 127–138.