Knowledge Base Reformation: Preparing First-Order Theories for Efficient Propositional Reasoning*

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

Gerhard Schurz
Department of Philosophy, Section Logic and Philosophy of Science
University of Salzburg
Franziskanergasse 1, A-5020 Salzburg, Austria
Email: gerhard.schurz@sbg.ac.at

Keywords. knowledge representation, knowledge compilation, automated reasoning, complexity of reasoning, propositional reasoning, relevance reasoning, transformation of logic programs

Abstract

We present an approach to knowledge compilation that transforms a function-free first-order Horn knowledge base to propositional logic. This form of compilation is important since the most efficient reasoning methods are defined for propositional logic, while knowledge is most conveniently expressed within a first-order language. To obtain compact propositional representations, we employ techniques from (ir)relevance reasoning as well as theory transformation via unfold/fold transformations. Application areas include diagnosis, planning, and vision. Preliminary experiments with a hypothetical reasoner indicate that our method may yield significant speedups.

*This paper is an improved and significantly extended version of Prendinger and Ishizuka [22].

1 Introduction

The need for knowledge base (KB) reformation derives from two facts about declarative representations of knowledge. First, representations are designed for variety of queries; hence, they will likely contain information that is not relevant to answering some particular query or query type. Second, many interesting problems in artificial intelligence require the representational power and compactness of first-order theories, but it is well-known that reasoning with such theories is computationally expensive (e.g. Papadimitriou [18]). 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 (e.g. Selman and Kautz [26]); NBP is a fast mechanism for solving propositional hypothetical (or abductive) reasoning problems (Ohsawa and Ishizuka [17]). The aim of KB 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 KB.

KB reformation extends existing work on *knowledge compilation* (Cadoli and Donini [2], Williams and Nayak [29]) to the first-order case. Compilation methods preprocess a propositional KB off-line such that the result can be used to speed up on-line query answering. By contrast, we start with a function-free and non-recursive first-order Horn theory and generate a propositional Horn theory of manageable size (which might then be further preprocessed by propositional compilation methods).

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 instantiation. The problem of *economically* instantiating clauses will be called the 'instantiation problem'. In order to obtain a knowledge base of practical size, we operate both on the level of clauses and the level of instantiations (of clauses). In particular, we use methods from the following fields:

- **Relevance reasoning.** Methods from this area are used to reduce the number of clauses considered to answer some instance of a query type (Levy *et al.* [13], Schurz [25]). Moreover, relevance reasoning is employed to instantiate the theory.
- **Theory transformation.** A principled application of unfold/fold

rules allows to reduce the number of possible instantiations of a clause. More specifically, theory transformation eliminates 'unnecessary' variables, i.e., variables that occur in the body B but not in the head H of a clause $H \leftarrow B$ (Tamaki and Sato [28], Proietti and Pettorossi [24]).

Our approach is clearly inspired by the success of the 'planning as satisfiability' approach of Kautz and Selman [10], where first-order *planning problems* are 'encoded' as propositional satisfiability problems. To obtain an efficient propositional representation, the encoding exploits specific assumptions about the planning domain. By contrast, we propose a *general theory* for representing first-order Horn theories in propositional logic, i.e., we propose methods to transform *any* first-order Horn theory to a propositional one, thus encompassing problems different from planning, such as diagnosis or vision.

The advantages of our KB reformation framework are as follows. First, our framework is applicable to a wide range of first-order Horn theories. Second, the propositional theories resulting from KB reformation have attractive computational properties such as small size and shorter clauses. Third, the propositional theories can be derived *automatically* from their first-order pendants. Consequently, researchers interested in propositional algorithms may approach problems traditionally formulated in first-order Horn logic with little extra effort.

The main contribution of this paper is an effective compilation method transforming first-order Horn theories to propositional Horn theories, based on the structure-sensitive application of relevance reasoning in conjunction with a novel set of variable elimination procedures.

The paper is organized as follows. In Section 2, we show how techniques from relevance reasoning can be used to rule out parts of a knowledge base that are not related to a set of queries. Section 3 offers a solution to the instantiation problem, by means of procedures that eliminate unnecessary variables from clauses. In Section 4, 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 preliminary experimental results. In Section 6, we discuss the paper and related work, and in Section 7, we give some conclusions.

2 Relevance Reasoning

In this section, we will introduce procedures that first partition a Horn theory into (possibly independent) subtheories, and then remove clauses

from a subtheory that cannot contribute to the solution of any query, also called *strongly (proof-based) irrelevant* clauses (e.g. Levy *et al.* [13], Schurz [25]). The resulting theories significantly reduce the search space for logical reasoners.

2.1 Preliminaries

We start with some general definitions from logic programming (e.g. Lloyd [14]). 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 $X_{i,1}, ..., X_{i,m_i}$. If $n > 0$ then C is called a *rule* (or *non-unit* clause), else ($n = 0$) C is called a *fact* (or *unit* clause). 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). If the head of a clause C does not occur elsewhere in the body of some clause in T, C is called a *definition* clause.

Horn clauses considered are subject to the following restrictions.

- *Function-free.* A clause C is function-free if it does not contain non-zero arity function symbols. On the other hand, zero-arity function symbols (constants) are allowed.

- *Range-restricted.* 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 $\mathcal{V}(hd(C)) \subseteq \mathcal{V}(bd(C))$. For instance, $p(X) \leftarrow q(X)$ is range-restricted, whereas $p(X, Y) \leftarrow q(X)$ is not.

All clauses considered here are Horn, function-free, and range-restricted.

The *Herbrand base* $T_{\mathcal{B}}$ of a theory T is the set of all ground instances of atoms formed by predicates and constants in T (a new constant is invented if T does not contain any constants). We impose the following restriction on theories.

- *Acyclic [1].* A theory T is acyclic if there is an assignment of a positive integer to each element in the Herbrand base $T_{\mathcal{B}}$ of T such that for every clause $C_{\mathcal{B}}$ in $T_{\mathcal{B}}$, the number assigned to the atom in $hd(C_{\mathcal{B}})$ is greater than the number of assigned to each atom in $bd(C_{\mathcal{B}})$.

More conspicuously, a theory T is acyclic if the (disjunct) directed graphs corresponding to $T_{\mathcal{B}}$ contain no cycles. Observe that if all clauses in a theory T are function-free and T is acyclic, then there cannot be recursive definitions in T. In general, however, acyclicity does not preclude recursive definitions (see Apt and Bezem [1]). An interesting subclass of acyclic theories are tree-structured theories. A theory T is called *tree-structured* if the directed graph corresponding to T consists of subtrees $T_1, ..., T_n$, i.e., each T_i has only one top node and there exists only one (directed) path from every node in T_i to the top node.

2.2 Theory Factorizing

The idea of *theory factorizing* is to remove all clauses that cannot contribute to answering some query type $p(\bar{X})$. We define factorizing w.r.t. query *types* $p(\bar{X})$ rather than particular instantiations of queries such as $p(a)$ or $p(b)$. Consequently, we only consider the predicate symbols occurring in a clause, and sometimes write query types $p(\bar{X})$ simply as p. Note that we have to assume that every distinct predicate symbol has a unique arity; for instance, $p(X)$ and $p(X, Y)$ do *not* refer to the same relation.

The theory factorizing procedure comes in two versions, depending on the topology (structure) of the knowledge base.

Acyclic theories. If the theory is acyclic, factorizing is performed by means of an algorithm that computes all clauses that are 'reachable' from a query type p. The following definition relies on the standard conception of queries as clauses of the form $\leftarrow p$.

Definition 2.1
Let $\mathcal{C}$ be a set of clauses and p an atom occurring in the body of a clause C of $\mathcal{C}$. The set of atoms *reachable from* p is defined inductively as follows:

1. p is reachable from p if p occurs in the head of a clause $C' \in \mathcal{C}$ where $C \neq C'$.

2. Let $q \leftarrow p_1 \wedge ... \wedge p_n$ be a clause C of $\mathcal{C}$ and q an atom reachable from p. Then any atom p_i $(1 \leq i \leq n)$ occurring as head in a clause $C' \in \mathcal{C}$ is reachable from p if $C \neq C'$. The atom q is called an *ancestor* of each atom p_i. Each p_i is a *successor* of q.

The notion of reachability can be extended to non-unit clauses (rules) as follows. A non-unit clause C is reachable from an atom p if the the head of C is reachable from p. The set of clauses reachable from p is denoted by T_p. It

function $factorize(T)$ **return** a partition $\mathcal{P}$ of T
Input theory T and $\mathcal{P} = \{\{Q\}\}$, where Q is an atomic query type.

- { **for each** $C \in T$
  where C resolves with $\mathcal{D}_1, ..., \mathcal{D}_k \in \mathcal{P}$
  - **if** $k = 0$ **then** $\mathcal{P} = \mathcal{P} \cup \{\{C\}\}$;
  - **if** $k \geq 1$ **then** $\mathcal{P} = (\mathcal{P} \backslash \{\mathcal{D}_1, ..., \mathcal{D}_k\}) \cup \{\mathcal{D}_1 \cup ... \cup \mathcal{D}_k \cup \{C\}\}$;
  - $T = T \backslash \{C\}$;
- **return** $\mathcal{P}$ }

Figure 1: THEORY FACTORIZING **Procedure**.

is well known that the *reachability problem* is polynomial (e.g. Papadimitriou [18]).

Tree-structured theories. If the theory T is tree-structured, T can be 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 subtheory T_j, if $i \neq j$. This means that the search space for a given atomic query type $p(\bar{X})$ that is admissible in T_i can be restricted to a single subtheory T_i. A query type $p(\bar{X})$ is called *admissible in* T_i if it resolves with some clause in T_i. A polynomial algorithm for theory factorizing is given in Fig. 1.

Basically, 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 element of the 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 element of the partition while the old elements $\mathcal{D}_1, ..., \mathcal{D}_k$ get cancelled. The procedure halts when all clauses of the original clause set T are processed.

Example 2.1
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) $s1(a,b)$.

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.

$$T_1 = \{\leftarrow p; p \leftarrow q; p \leftarrow r; q \leftarrow q1 \wedge h\_1; r \leftarrow h\_2; r \leftarrow h\_3\}$$
$$T_2 = \{s \leftarrow s1; s1\}$$
$$T_3 = \{t \leftarrow h\_4\}$$

For query type p, the independent subtheories T_2 and T_3 are strongly irrelevant. In the case of an atomic query type p, at most one subtheory, which is denoted by T_p, is relevant to p. If the query type is non-atomic, for instance, $p \wedge q$, two subtheories have to be considered unless p and q resolve with some clauses in the same subtheory.

The following theorem summarizes the effect of theory factorizing. It can be easily derived from Th. 7 in Schurz [25].

Theorem 2.1
Let T be a first-order Horn theory, and $p(\bar{X})$ an admissible query type in T.

- *If T is acyclic, then any instance of $p(\bar{X})$ can be answered by only considering the set T_p. The clauses in $T \backslash T_p$ are strongly irrelevant to any instance of $p(\bar{X})$ w.r.t. T.*
- *Suppose T is tree-structured and $\mathcal{P} = \{T_1, ..., T_n\}$ the partition of T. Then any instance of $p(\bar{X})$ can be answered by considering exactly one independent subtheory $T_i \in \mathcal{P}$. All theories T_j with $i \neq j$ are strongly irrelevant to any instance of $p(\bar{X})$ w.r.t. T.*

Theory factorizing provides *necessary* conditions for strong irrelevance, i.e., given a query type $p(\bar{X})$, when a clause C is removed from T then C is strongly irrelevant to $p(\bar{X})$ w.r.t. T. In general, those procedures do not give sufficient conditions for strong irrelevance: after factorizing, the theory may still contain strongly irrelevant clauses. This is a consequence of ignoring particular instantiations of atoms. To see this, consider the following example.

Example 2.2
Assume the theory $T = \{p(X) \leftarrow q(X); q(b)\}$, and the query is $p(a)$. The clauses in T are not part of any proof of $p(a)$, although both clauses are output by factorizing. In fact, there exists no proof for $p(a)$.

function $simplify(T)$ **return** simplified theory T'
Input initial theory T^0, $ng(T^0) = \{q_1, ..., q_n\}$.

- { **repeat**

  { **for every** $q \in ng(T^i)$
  - delete all clauses $C_1, ..., C_m$ with q in the body **and**
  - delete all clauses $D_1, ..., D_l$ reachable from clauses $C_1, ..., C_m$;
  - $T^{i+1} = T^i \backslash \{C_1, ..., C_m, D_1, ..., D_l\}$;

- **until** $ng(T^k) = \emptyset$ for some $k > 0$ };

- **return** T^k }

Figure 2: THEORY SIMPLIFICATION **Procedure**.

2.3 Theory Simplification

A given (independent) theory may still contain strongly irrelevant clauses after factorizing. This is the case when a clause C contains an atom in $bd(C)$ that does not resolve with the head of any other clause. C is called a *failing* clause. Essentially, *theory simplification* removes failing clauses from a theory.

Assume we are given a theory T. The predicate symbols in T, $\mathcal{P}(T)$, can be divided into (i) *goal* predicates, i.e., predicate symbols that occur in the head of some clause in T (and possibly in the body of some clause), and (ii) *non-goal* predicates, that is, predicate symbols that *only* occur in the body of some clause in T. The set of goal predicate symbols is denoted by $g(T)$ and the set non-goal predicate symbols is denoted by $ng(T)$. Note that $g(T) \cap ng(T) = \emptyset$ and $g(T) \cup ng(T) = \mathcal{P}(T)$. For generality, we also allow for *assumable* predicates $\mathcal{H} \subseteq g(T)$. Atoms with assumable predicates denote hypotheses that *may* contribute to the proof of a query, if assumed as hypotheses. Because of their importance for hypothetical reasoning (e.g. Ohsawa and Ishizuka [17]), we want to consider them in the simplification procedure.

Simplification means that clauses C containing an atom $p \in ng(T)$ can be deleted from the theory T, producing the simplified theory T'. The simplification process is repeated until no failing clauses are detected. The theory simplification algorithm is presented in Fig. 2.

Theory simplification is best explained by continuing Example 2.1.

Example 2.3
Let T_1 be our initial theory

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

where strings starting with "$h\_$" denote hypotheses. Here, $g(T_1) = \{p, q, r, h\_1, h\_2, h\_3\}$ and $ng(T_1) = \{q1\}$. Since $q1$ does not resolve with any clause, the clause $q \leftarrow q1 \wedge h\_1$ can be deleted from T_1, resulting in T_1' where $ng(T_1') = \{q\}$. After the clause $p \leftarrow q$ is deleted, the remaining theory is

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

where $ng(T'') = \emptyset$. As each simplification step reduces the number of clauses in the (finite) theory, the simplification procedure will eventually halt.

The following theorem extends Th. 7 in Schurz [25].

Theorem 2.2
Let T be a first-order Horn theory, C a clause in T, and $p(\bar{X})$ an admissible query type in T. If C is removed by theory simplification, then C is strongly irrelevant to any instance of $p(\bar{X})$ w.r.t. T.

Similar to theory factorizing, theory simplification provides necessary but not sufficient conditions for strong irrelevance (see Example 2.2 for an example).

This concludes the first reformation phase. Although the procedures introduced so far reduce the number of clauses that have to be considered when answering a query instance, they do not target the instantiation problem. In the next section, we will propose a solution to the instantiation problem. After that, we will return to issues of relevance again. Below, we discuss some practical implications of KB reformation by relevance reasoning.

2.4 Practical Implications of Relevance Reasoning

It is well known that a major factor of slowing down reasoning mechanisms for artificial intelligence is that parts of the knowledge base irrelevant to the query are explored (Levy *et al.* [13], Darwiche [4]). We consider the application field of *model-based diagnosis* (e.g. de Kleer *et al.* [6]).

If the system description (behavioral model) is tree-structured, the diagnostic task is decomposable and allows for highly efficient diagnosis procedures (see Stumptner and Wotawa [27]). From a KB reformation point of

view, tree-structured systems are advantageous since the theory factorizing procedure has to be applied only once. On the other hand, if the system description is acyclic, factorizing has to be invoked $2^n - 1$ times, where n is the number of possible observations (manifestations) about the system behavior. In other words, we have to determine the relevant portion of the system for each possible subset of the set of atomic observations (excluding the case where no observation is made). In practice, however, it is often sufficient to compute the relevant portion of the model for observation sets of small cardinality.

The factorizing procedure developed for tree-structured theories can also be applied to acyclic theories, although factorizing via the reachability algorithm generally yields smaller subtheories. To see this, observe that according to the reachability algorithm, only successors p of a query type q enter the subtheory, whereas a partition generated by theory factorizing also contains the ancestors r of successors p of the query type q, where r does not contribute to a proof of the query in question. For instance, let the theory be $T = \{s_1 \leftarrow s_0; s_2 \leftarrow s_0; s_0\}$, with query s_1. The reachability procedure generates the set $\{\leftarrow s_1; s_1 \leftarrow s_0; s_0\}$, whereas theory factorizing yields T as the single subtheory because s_0 and $s_2 \leftarrow s_0$ resolve.

The practical value of theory simplification consists in its ability to detect unspecified context conditions in a system description. A situation where a subquery does not resolve with a fact indicates that some input to a device has not been declared in the description.

In Prendinger and Ishizuka [23, 21], KB reformation by relevance reasoning is proposed as a compilation step preceding the actual diagnosis. Empirical results indicate the potential of relevance reasoning for efficient diagnosis.

3 Theory Transformation

The motivation for theory transformation by applying unfold/fold transformations is to reduce the complexity of a first-order Horn theory *as measured by the number of possible ground instantiations* of clauses. Unfold/fold transformations were originally introduced by Tamaki and Sato [28] to enhance the efficiency of logic programs. Kawamura and Kanamori [11] show the equivalence-preserving nature of the transformation rules. A precise definition of the transformation rules is given in the Appendix.

We will propose theory transformation to solve the 'instantiation problem', i.e., the problem that simple-minded instantiation of variables by con-

stants yields a propositional theory of impractical size. Since the instantiation of a clause C is exponential in the number of different variables occurring in C, we try to minimize that number. Consider the clause C_0 with four different variables (X, Y, $Z1$, and $Z2$)

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

that yields $\mathcal{O}(n^4)$ clauses upon instantiation. Theory 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. Here, C_2 is introduced by the definition rule and then folded with C_0, yielding C_1. To demonstrate the instantiation problem by means of a concrete example, 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_0, and 125 possibilities for each of C_1 and C_2. Hence the original theory $T = \{C_0\}$ allows for 625 propositional clauses, whereas the reformed theory $T' = \{C_1, C_2\}$ has only 250 instantiations.

More specifically, we will eliminate *unnecessary* variables, i.e., variables that occur in the body $bd(C)$ but not in the head $hd(C)$ of a clause C. According to this definition, the variables $Z1$ and $Z2$ in C_0 are unnecessary.

Definition 3.1
A first-order Horn theory T is called *optimally reduced* if no clause in T contains unnecessary variables.

Obviously, the aim of theory transformation (as motivated here) will be to generate optimally reduced theories. Indeed, one algorithm described in Proietti and Pettorossi [24] eliminates *all* unnecessary variables as a result of the sequential application of unfolding, definition and folding steps. Unfortunately, the algorithm only terminates for very specific theories. More precisely, the procedures halt for theories that can be split into nonascending and tree-like theories (see Proietti and Pettorossi [24] for an explanation of this rather intriguing property). The non-halting property undercuts every attempt to develop *algorithmic* transformation procedures. We will propose procedures that halt on *every* theory, but do not necessarily eliminate all unnecessary variables.

3.1 Definitions

We start with some terminology to distinguish different kinds of clause bodies. The distinction is intended to cover a broad range of syntactically possible clause bodies. The most central notion is that of a *block* of a clause body, which we borrow from Proietti and Pettorossi [24].

Definition 3.2 (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 w.r.t. R^*. Note that each (distinct) 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 $\{p2(Y, Y1)\}$.

Definition 3.3 (faithful variant)
A block Bl_1 in the body of a clause C_1 is called a *faithful variant* of a block Bl_2 in the body of a clause C_2 if and only if there exists a renaming substitution θ such that:

- $Bl_1 = Bl_2\theta$ (Bl_1 is a variant of Bl_2), and
- for all X in $\mathcal{V}(Bl_2)$, X is an existential variable in C_2 if and only if $X\theta$ is an existential variable in C_1.

A block can have a variety of syntactical forms. For clause bodies that contain blocks or fall under one of the following definitions, we will later propose procedures that eliminate unnecessary variables.

Definition 3.4 (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

- for all adjacent A_i, A_j, we have $R(A_i, A_j)$;
- the first and the last atom in the (reordered) block contain at least one variable occurring in $hd(C)$;

- no other atom contains a variable occurring in $hd(C)$;
- the variables in $R(A_i, A_j)$ are distinct.
- all variables in each $R(A_i, A_j)$ are distinct from the variables in $hd(C)$.

Then $Bl = \{\langle A_1, ..., A_k \rangle\}$ is called a *chain*.

Reconsider clause C_0,

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

where the $bd(C_0)$ is a single block that forms a chain. The atoms $p1(X, Z1)$ and $p3(Z2, Y)$ are called *opening* and *closing* atoms, respectively. Assume a chain where A_1 is the opening atom and A_k is the closing atom. Variables such that $X \notin (\mathcal{V}(A_1) \cap \mathcal{V}(A_2)) \cup (\mathcal{V}(A_{k-1}) \cap \mathcal{V}(A_k))$ are called *embracing* variables of A_1 and A_k. Let $\langle p1(X, Z1), p2(Z1, Z2), p3(Z2, Y) \rangle$ be a chain. Here X and Y are the embracing variables.

Definition 3.5 (loop)
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 3.6 (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 $X \subseteq \mathcal{V}(B_i)$: all elements in X occur only once in $bd(C)$. 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.

3.2 Variable Elimination Procedures

The following procedures automatize the definition rule for clause bodies defined in the previous subsection. The resulting clauses are then folded with the original clauses. Thereby we usually obtain slightly more first-order clauses, but they yield significantly fewer instantiations.

Procedure 3.1 (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. 3.2). 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 $newp/m$ is a fresh predicate symbol and $\{Y_1, ..., Y_m\}$ is defined as $(\mathcal{V}(B_i) \cup \mathcal{V}(B_j)) \setminus (\mathcal{V}(B_i) \cap \mathcal{V}(B_j))$, else (only one unnecessary variable) do nothing. Note that this procedure is non-deterministic, since $R(B_i, B_j)$ is not uniquely determined.

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, and on backtracking, the clause

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

is generated.

Procedure 3.2 (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

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

where $newp/n$ is a fresh predicate symbol and $X_1, ..., X_n$ are the embracing variables of A_1, A_{k-1}. An example will be described below.

Procedure 3.3 (isolated blockpart)
Let C be a clause with $p(X_1, ..., X_n)$ $(n \geq 1)$ an atom in a block Bl in $bd(C)$. Suppose $\bar{Y}$ is the set of isolated variables in Bl and $\bar{X}' = \{X_1, ..., X_n\} \setminus \bar{Y}$. 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 is described in Fig. 3.[1] Note that in the last step after all unfolding—definition—folding cycles, all non-definition clauses are deleted, i.e., the clauses that have been used for unfolding (and are not needed any more).

In the next subsection, we will go through the algorithm by means of an example. The procedure significantly extends the algorithm in Proietti and Pettorossi [24]). Except for a special instance of clause bodies with isolated blockparts, their algorithm did not eliminate any unnecessary variables.

To summarize, in the second phase of KB reformation, a theory T is transformed to an equivalent theory T' that allows for significantly fewer instantiations.

3.3 Example

In this subsection, we describe an example for a subset of the variable elimination procedures in detail. Let the theory T_{path} consist of the following single clause.

$$(r_0)\ \ path(X,Y) \leftarrow link1(X,Z1) \wedge link2(Z1,Z2) \wedge link3(Z2,Z3) \wedge link4(Z3,Y)$$

The predicates $link1$ – $link4$ can be assumed as hypotheses. The first step in the transformation sequence starting from T_{path} is an application of the definition rule. Observe that r_0 (trivially) cannot be unfolded with any clause because it is the only clause in T_{path}. Since the body of r_0 forms a chain, we apply Proc. 3.2 and obtain the clause

$$(r_1)\ \ newp1(X,Z3) \leftarrow link1(X,Z1) \wedge link2(Z1,Z2) \wedge link3(Z2,Z3)$$

Then a folding step is performed, with r_0 as the folded clause and r_1 as the folding clause, yielding

$$(r_2)\ \ path(X,Y) \leftarrow newp1(X,Z3) \wedge link4(Z2,Z3)$$

[1] Recall that the transformation rules are explained in the Appendix.

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

For each definition clause $D \in \mathcal{D}$ that n contains unnecessary variables ($n > 0$) **do** {

1. **Unfolding step:** unfold some atom in the body of D using non-unit clauses $F_1, ..., F_n$ in T and put the resulting clauses into a set $U_D = \{E_1, ..., E_n\}$;

2. **Definition steps: for each** clause $E_i \in U_D$

   for each block $Bl \in partbd(E_i)$ where

   - Bl contains at least one unnecessary variable, and
   - Bl is not a faithful variant of the body of any clause in $\mathcal{D} \cup \mathcal{PD}$,

   do {

   - **if** Bl is a chain, **apply** Proc. 3.2,
   - **if** Bl contains an isolated blockpart, **apply** Proc. 3.3,
   - **else apply** Proc. 3.1;

   and add the new definition rule to $\mathcal{D}$ };

3. **Folding steps: for each** clause E_i in U_D add to T' the clause resulting from E_i as follows:

   for every block Bl of $partbd(E_i)$ which is a faithful variant of a body of a clause G in $\mathcal{D} \cup \mathcal{PD}$, fold Bl in E_i using G.

$\mathcal{D} = \mathcal{D} \backslash \{D\}$, $\mathcal{PD} = \mathcal{PD} \cup \{D\}$, $\mathcal{P} = \mathcal{P} \cup \{F_1, ..., F_n\}$ }
$T' = T' \backslash \mathcal{P}$

Figure 3: Theory Transformation **Procedure**.

# constants	# clauses T_{path}	# clauses T_{ref}
2	32	24
3	243	81
4	1024	192
5	3125	375

Figure 4: Comparison of the size of original and reformed theories.

The next definition rule to deal with is r_1. Its body has the form of a chain, hence Proc. 3.2 is applied to generate clause

$$(r_3) \quad newp2(X, Z2) \leftarrow link1(X, Z1) \wedge link2(Z1, Z2)$$

A folding step follows, with r_1 as the folded clause and r_3 as the folding clause, producing

$$(r_4) \quad newp1(X, Z3) \leftarrow newp2(X, Z2) \wedge link3(Z2, Z3)$$

As a result, r_0 is equivalently replaced by

$$(r_2) \quad path(X, Y) \leftarrow newp1(X, Z3) \wedge link4(Z3, Y)$$
$$(r_4) \quad newp1(X, Z3) \leftarrow newp2(X, Z2) \wedge link3(Z2, Z3)$$
$$(r_3) \quad newp2(X, Z2) \leftarrow link1(X, Z1) \wedge link2(Z1, Z2)$$

after theory transformation. In Fig. 4, the size of the instantiated theory $T_{path} = \{r_0\}$ is compared to that of the reformed theory $T_{ref} = \{r_2, r_3, r_4\}$, for 2 to 5 constants.

3.4 A Note on Recursive Theories

The definition of a chain seems particularly interesting, since it can be used to encode certain forms of *recursive* theories, such as

$$(1) \quad path(X, Y) \leftarrow link(X, Y)$$
$$(2) \quad path(X, Y) \leftarrow link(X, Z) \wedge path(Z, Y)$$

If the length of the longest path can be determined, say as n, we may replace the recursive definition (2) by clauses with chain bodies of the form $link_1(X, Z_1) \wedge ... \wedge link_m(Z_{m-1}, Y)$ for all $1 \leq m \leq n$. However, the set of (almost) all initial subsequences of the maximal chain can also be obtained by Proc. 3.2. In the example above, the newly introduced predicates $newp1$ and $newp2$ cover paths of length 3 and 2, respectively. The halting condition (1) has to be added separately. It has the form $path(X, Y) \leftarrow link_i(X, Y)$ $(1 \leq i \leq n)$.

4 Relevance Reasoning Revisited

In the last phase of the KB reformation process, the theory is actually instantiated. By employing the query-tree idea of Levy *et al.* [13], 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 w.r.t. the theory. In our approach, the query tree idea is applied in a novel way, where the (second phase of the) construction of the query-tree is used to economically instantiate the theory. Our instantiation procedure extends the *intelligent grounding module* (IG) proposed for the Disjunctive Deductive Database System `dlv` (Eiter *et al.* [7]). However, our procedure only applies to non-disjunctive theories.

4.1 Levy's Query-Tree

A *query-tree* is a compact representation of a search tree for first-order Horn theories (Levy *et al.* [13]). Most importantly, the query tree encodes precisely the set of all derivations of a query type $p(\bar{X})$. In brief, the query-tree is an AND-OR tree with goal-nodes and rule-nodes (more detail will be given below). Since we do not allow for recursion in clauses, our construction of the query tree is simpler than the original one in [13]. 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 [17].

Example 4.1
Consider the following theory (the atom with predicate h denotes a hypothesis).

$$
\begin{array}{lll}
(r_1)\ \ p(X,Y) \leftarrow q1(X,Y) \wedge q2(X,Y). & & \\
(r_2)\ \ q1(X,Y) \leftarrow r1(X,Y) \wedge h(X,Y). & & \\
(r_3)\ \ q2(X,Y) \leftarrow r2(X,Y). & & \\
(r_4)\ \ q2(X,Y) \leftarrow r3(X,Y). & & \\
(f_1)\ \ r1(a,b). & (f_2)\ \ r1(a,c). & \\
(f_3)\ \ r2(a,b). & (f_4)\ \ r2(c,d). & (f_5)\ \ r3(b,d).
\end{array}
$$

The query tree algorithm consists of two phases.

Bottom-up phase. A set of *adorned* predicates and rules is generated. An adorned predicate p^c is a predicate p with constraint c on its arguments. The adorned rules are the rules of the theory T with predicates replaced by adorned predicates. We start with the base predicates of the T. Base predicates denote relations that have no definition in T, i.e., the predicates

of facts and hypotheses. For instance, the adorned predicate $r1^c$ is obtained by completion (Clark [3]):

$$r1(X,Y) \leftrightarrow (X = a \wedge Y = b) \vee (X = a \wedge Y = c).$$

For convenience, the adornment of $r1$ is written as $c(X,Y) = \{\langle a,b\rangle, \langle a,c\rangle\}$. Let U be the set of all constants appearing in T. Then the adornment of the (uninstantiated) hypothesis h in T is $\{\langle X,Y\rangle : \langle X,Y\rangle \in U^2\}$. Given a rule in T of the form $q(\bar{X}_{n+1}) \leftarrow p_1(\bar{X}_1) \wedge ... \wedge p_n(\bar{X}_n)$ and an adorned predicate $p_i^{c_i}$ for each predicate p_i. If $c = c_1(\bar{X}_1) \wedge ... \wedge c_n(\bar{X}_n)$ is satisfiable, we add the adorned rule

$$q^{c_h}(\bar{X}_{n+1}) \leftarrow c \wedge p_1^{c_1}(\bar{X}_1) \wedge ... \wedge p_n^{c_n}(\bar{X}_n)$$

where q^{c_h} is the projection of c on the head variables $\bar{X}_{n+1}$. The bottom-up phase terminates when no new adornments are generated. In Example 4.1, the following predicate adornments are generated.

$r1^{\{\langle a,b\rangle,\langle a,c\rangle\}}$ $\quad$ $r2^{\{\langle a,b\rangle,\langle c,d\rangle\}}$ $\quad$ $r3^{\{\langle b,d\rangle\}}$ $\quad$ $h^{\{\langle a,a\rangle,\langle a,b\rangle,...\}}$
$p^{\{\langle a,b\rangle\}}$ $\quad$ $q1^{\{\langle a,b\rangle,\langle a,c\rangle\}}$ $\quad$ $q2^{\{\langle a,b\rangle,\langle c,d\rangle,\langle b,d\rangle\}}$

Top-down phase. Starting with the node of the adorned query type q^c, we construct the query-tree such that each node g of a predicate p has a *label* $l(g)$. Initially the goal-node $l(g) = q^c$ is created (see Fig. 5). A goal-node g for a predicate q^{c_h} can be unified with adorned rules r of the form $q^{c_h}(\bar{X}_{n+1}) \leftarrow c \wedge p_1^{c_1}(\bar{X}_1) \wedge ... \wedge p_n^{c_n}(\bar{X}_n)$. If $l(g) \wedge c$ is satisfiable, a rule-node g_r is created as a child of g, with $l(g_r) = l(g) \wedge c$ as its label. For every body atom $p_i^{c_i}$ in r, the rule-node $l(g_r)$ has a child goal-node whose label is the projection of $l(g_r)$ onto $\bar{X}_i$. Since nodes of base predicates and nodes with unsatisfiable label, denoted by $l(-)$, are not expanded, the top-down construction halts.

Levy *et al.* [13] show that the complexity of building the query-tree is linear in the number of rules and possibly exponential in the arity of predicates. The following theorem is due to Levy *et al.* [13].

Theorem 4.1
Let T be a first-order Horn theory, and $p(\bar{X})$ an admissible query type in T. Let $\mathcal{T}$ be the query-tree generated from the clauses in T.

- *A fact $p(a_1, ..., a_n)$ is strongly irrelevant to any instance of $q(\bar{X})$ w.r.t. T if and only if there is no node g of p in $\mathcal{T}$ such that $a_1, ..., a_n$ satisfies the label $l(g)$ of g.*

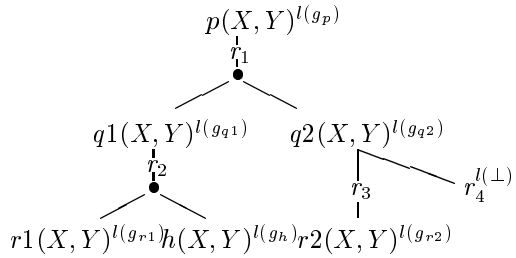

Figure 5: Query-tree for Example 4.1. The label for each goal-node g is $l(g) = \{\langle a, b\rangle\}$. For simplicity, labels of rule-nodes are omitted. Note that expanding node $q2(X, Y)$ with rule r_4 would result in an inconsistent label.

- *A rule r is strongly irrelevant to any instance of $q(\bar{X})$ w.r.t. T if and only if r does not appear in $\mathcal{T}$.*

Relevance reasoning by Levy's query-tree provides necessary *and sufficient* conditions for strong irrelevance, a fact that will be exploited in the next subsection.

4.2 Instantiation

In our approach, instantiation of variables to constants will be performed as a by-product of the top-down construction of the query-tree. More specifically, if $l(g_r)$ is the label of a rule r in the query-tree, the propositional version of r is obtained by performing all unifications appearing $l(g_r)$. Hence the output for Example 4.1 is the following theory.

(r'_1) $p(a,b) \leftarrow q1(a,b) \wedge q2(a,b).$
(r'_2) $q1(a,b) \leftarrow r1(a,b) \wedge h(a,b).$
(r'_3) $q2(a,b) \leftarrow r2(a,b).$
(f'_1) $r1(a,b).$ $\quad$ (f'_3) $r2(a,b).$

It is a consequence of the construction of the query-tree that $p(a, b)$ is the only instance of the query type $p(X, Y)$ which has a solution (given that $h(a, b)$ is assumed). As output of the reformation procedure, we obtain propositional theories indexed with query types.

5 Preliminary Empirical Evaluation

Hypothetical reasoning. The impact of the savings gained by KB reformation is tested by means of the Networked Bubble Propagation (NBP)

mechanism [17], an efficient *propositional hypothetical reasoning* method for computing near-optimal solutions (e.g. diagnoses). In hypothetical reasoning, we are given a knowledge base T, hypotheses $\mathcal{H}$, and a query q. Sometimes the problem formulation contains inconsistency constraints $I \subset T$ of the form "$inc \leftarrow h_1(\bar{t}_1) \wedge ... \wedge h_n(\bar{t}_n)$" where $h_i \in \mathcal{H}$, $\bar{t}_i$ a sequence of constant symbols, and the symbol "inc" denotes the impossible state.[2] 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) $T \cup H_i \nvdash inc$.

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 (e.g. Poole [20]). 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.

For the experiments we use a Sun Ultra 2 workstation with 320 MB memory. The KB reformation methods are implemented in Sicstus Prolog, the NBP method is implemented in C. All reformation procedures except theory transformation (which requires the user to select atoms for unfolding) are automatic. Running times exclude the time needed for KB reformation. For example, the relevant part of a theory consisting of 1000 clauses can be extracted in about 4 seconds.

Acyclic Horn theories. The first experiment is intended to show the efficiency gain of theory factorizing and simplification.[3] It involves first-order Horn theories from 40 up to 1000 rules with a fixed percentage of integrity constraints (about 20%) and few facts. The theory of Ex. # 1 is systematically expanded to theories of larger size. Fig. 6 shows the inference time as a function of the number of hypotheses before and after reformation by relevance reasoning (the same solution sets H are generated). Results are obtained by averaging over three different query types. The results show that the reformed theories can be processed more efficiently, usually in excess of a factor of 3, ranging up to 196 (for a query type where only 2% of the hypotheses are relevant).

[2] The reformation procedures discussed in this paper can be straightforwardly extended to account for integrity constraints.

[3] The speedup effect of the instantiation procedure was not tested in this experiment. We have to refer the reader to the experiments in [13].

Ex. #	# hypotheses		time (sec)	
	before ref.	after ref.	before ref.	after ref.
1	13	6	0.02	0.01
2	39	17	0.08	0.04
3	78	33	0.21	0.07
4	156	67	0.63	0.16
5	312	129	2.14	0.46

Figure 6: Comparison for acyclic Horn theories.

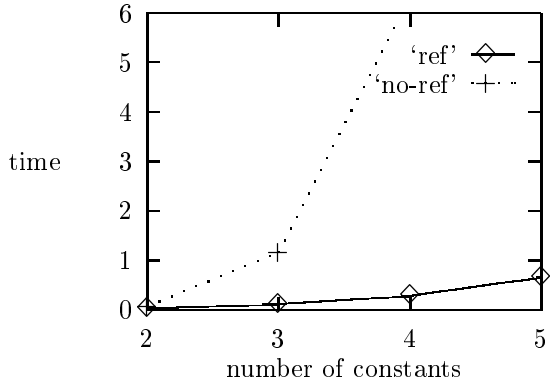

Figure 7: The effect of theory transformation in the path example. Time in seconds.

Path example. The second experiment was designed to show the speedup effect of KB reformation by theory transformation. We take the path example from Section 3 together with the inconsistency constraint "$inc \leftarrow link1(X,Y) \wedge link3(Y,X)$". All link-atoms can be assumed as hypotheses. Fig. 7 shows significant speedups for the reformed theories (which yield the same solution sets H as the original theories). Note that in this experiment, the number of hypotheses is constant but (i) the number processed rules is dramatically reduced, as shown in Fig. 4, and (ii) the reformed rules involve 2 instead of 4 body atoms. Unfortunately, we were not able to obtain results for unreformed path theories beyond 4 constants (1024 rules), whereas for reformed theories, we measured 2.7 sec for 7 constants (1029 rules). Observe that for almost the same number of rules, the reformed theory can be processed more than twice as fast as the original one (6.32 sec). We suspect that similar to satisfiability (SAT) problems, the number of propositional variables in clauses has a major impact on inference time.

At this stage of investigation, our experiments (not the results) are not completely satisfying. First of all, we want to test KB reformation on more diverse first-order Horn theories, and compare our reformation-based approach to other first-order hypothetical reasoning methods, e.g., Ng and Mooney [16]. Secondly, we seek to compare running times with a hypothetical reasoner that has better scaling properties than current NBP, for instance, the method of Ishizuka and Matsuo [8].

6 Discussion

Our approach to KB reformation might be seen as foundational work related to the 'planning as satisfiability' framework of Kautz and Selman [10], for the case of hypothetical reasoning with first-order Horn theories. Unlike the problem-specific *encodings* of planning problems discussed by Kautz *et al.* [9], we simply *represent* first-order problems in propositional logic. Our variable elimination procedures share some intuitions with the 'operator splitting' technique in Kautz and Selman [10], where, for instance, a 4-arity predicate $move(X,Y,Z,I)$ is replaced by three 2-arity predicates, $object(X,I)$, $source(Y,I)$, and $destination(Z,I)$. The purpose of both variable elimination and the 'operator splitting' technique is to obtain a small-sized theory upon instantiation.

Although we think that we discussed the major techniques for equivalent KB reformation, other methods might be of interest as well. For instance, Levy [12] proposes *theory abstraction* as a method to transform a given

theory to a (computationally) simpler one. The idea of theory abstraction is to remove some detail from the knowledge base, in particular, some *argument* of a predicate, if the argument is irrelevant to answering a given query type. An argument is said to be irrelevant (relative to a query) if (i) proving the query requires an arbitrary value for that argument, and (ii) the values for the argument are not subject to other constraints. Note that in this paper, clauses rather than predicate arguments were deleted from the knowledge base. The computational gain (for our present purpose) derives from the fact that rules containing less variables allow for less ground instantiations. We are just beginning to understand the merits of theory abstraction and leave the details for future research.

The proposed KB reformation procedures are guaranteed not to slow down inference and preserve all solutions, at the cost of a reasonable effort needed to generate the reformed knowledge base. Regarding relevance reasoning, Levy *et al.* [13] report on significant speedups with a deductive theorem prover. Our results complement those findings for a hypothetical reasoner. For instance, the effect of theory transformation is quite extreme in the path example. We realize, however, that in general, theories do not contain many unnecessary variables. Furthermore, we cannot provide guarantees that our variable elimination procedures cover all (or most) conceivable Horn theories. A prime task for future research is to evaluate the efficiency gain of KB reformation on more diverse Horn theories. We also plan to investigate syntactical properties that are computationally attractive and can be shown to be common to reformed theories. For the K-SAT problem (K the number of propositional variables in each disjunctive clause), Monasson *et al.* [15] show how the value of K affects computing time.

7 Conclusion

In this paper, we address the following problem: *given a problem formulation in function-free first-order Horn logic, how can we obtain a compact propositional representation?* This problem is important since the most efficient reasoning mechanisms are defined for propositional logic (e.g. [26, 17, 29, 5]). On the other hand, knowledge is most naturally represented in a first-order language.

In order to compile first-order theories to propositional theories, we employ techniques from diverse fields such as deductive databases (relevance reasoning) and logic programming (theory transformation). The theory factorizing procedures are sensitive to the structure (topology) of the knowl-

edge base. In the context of unfold/fold transformations, a set of special procedures is defined to eliminate unnecessary variables from clauses. The query-tree idea [13] is utilized in a novel way to generate the least number of instantiations of clauses.

Practical applications of our compilation paradigm include diagnosis, planning, and vision. Reformation-based diagnosis is discussed in [23].

Acknowledgments

In the first place, we would like to thank the anonymous referees for their very helpful and valuable comments. Special thanks go to Tetsu Yamamoto (University of Tokyo) who implemented the variable elimination procedures. The first author was supported by a fellowship from the Japan Society for the Promotion of Science (JSPS).

A Transformation Rules

Unfold/fold transformations for logic programs are first studied by Tamaki and Sato [28] (see also Pettorossi and Proietti [19] for a survey paper). The transformation rules are defined as follows.

Definition A.1 (definition rule)
The definition rule adds a *new* clause C of the form

$$newp(X_1, ..., X_k) \leftarrow A_1 \wedge ... \wedge A_n$$

to a given theory where $newp/k$ is a predicate symbol not contained in the theory and $A_1, ..., A_n$ are built from predicate symbols contained in the theory. The variables $X_1, ..., X_k$ are distinct variable symbols occurring in $A_1, ..., A_n$. The clause introduced by the definition rule is called a *definition* clause. The introduction of a definition clause is also called a *eureka step* and the predicate $newp/k$ a *eureka predicate*.

Definition A.2 (unfolding rule)
Given a Horn theory T_k, a clause C in T_k, and B an atom in its body; $C_1, ..., C_n$ are clauses in T_k such that the head of each C_i unifies with B, by unifiers $\theta_1, ..., \theta_n$ (note that unification is trivial since we do not admit function symbols). Let C_i' be the result of replacing B in C with the body of C_i followed by applying θ_i to the whole clause. Then $T_{k+1} = (T_k \backslash \{C\}) \cup \{C_1', ..., C_n'\}$. C is called the *unfolded* clause and $C_1, ..., C_n$ are called the *unfolding* clauses. B is called the *selected* atom.

The unfolding rule corresponds to applying resolution to clause C with A as the selected atom and $C_1, ..., C_n$ as input clauses.

Definition A.3 (folding rule)
Given a Horn theory T_k, and a clause C in T_k of the form

$$A \leftarrow B_1 \wedge ... \wedge B_n \quad (n > 0)$$

and a *new* clause D of the form

$$F \leftarrow E_1 \wedge ... \wedge E_m \qquad (m > 0)$$

Assume there exists a substitution θ such that

- $B_1 = E_1\theta, ..., B_m = E_m\theta$;
- for each variable X occurring only in $bd(D)$, θ substitutes a distinct variable not appearing in $\{F\theta, A, B_1, ..., B_n\} \backslash \{B_1, ..., B_m\}$;
- D is the only clause whose head is unifiable with $F\theta$.

Let C' be the clause

$$A \leftarrow F\theta \wedge B_{m+1} \wedge ... \wedge B_n$$

Then $T_{k+1} = (T_k \backslash \{C\}) \cup \{C'\}$. C is called the *folded clause* and D the *folding* clause.

Definition A.4 (transformation sequence)
Given T_0 as the initial theory, and T_{i+1} the result of applying a *transformation step* to T_i $(i \geq 0)$. A transformation step is an application of either the definition rule, or the folding rule or the unfolding rule. The sequence of theories $T_0, ..., T_n$ is called the *transformation sequence starting from* T_0.

References

[1] Krzysztof R. Apt and Marc Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.

[2] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10:137–150, 1997.

[3] Keith L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[4] Adnan Darwiche. A logical notion of conditional independence. properties and applications. *Artificial Intelligence*, 97(1–2):45–82, 1997.

[5] Adnan Darwiche. Compiling devices: a structure-based approach. In *Proceedings 6th International Conference on Knowledge Representation and Reasoning (KR-98)*, pages 156–166, 1998.

[6] J. de Kleer, A.K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2–3):197–222, 1992.

[7] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarello. The architecture of a disjunctive deductive database system. In *Proceedings Joint Conference on Declarative Programming (GULP-97)*, 1997.

[8] 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.

[9] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, 1996.

[10] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings 10th European Conference on Artificial Intelligence (ECAI-92)*, 1992.

[11] Tadashi Kawamura and Tadashi Kanamori. Preservation of stronger equivalence in unfold/fold logic program transformation. *Theoretical Computer Science*, 75:139–156, 1990.

[12] Alon Y. Levy. Creating abstractions using relevance reasoning. In *Proceedings 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 588–594, 1994.

[13] 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.

[14] John Wylie Lloyd. *Foundations of Logic Programming*. Springer, Berlin, New York, second, extended edition, 1987.

[15] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.

[16] Hwee Tou Ng and Raymond J. Mooney. Abductive plan recognition and diagnosis: A comprehensive empirical evaluation. In *Proceedings 3rd International Conference on Knowledge Representation and Reasoning (KR-92)*, pages 499–508, 1992.

[17] 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.

[18] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1995.

[19] Alberto Pettorossi and Maurizio Proietti. Transformation of logic programs. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (Volume 5, Logic Programming)*, pages 697–787. Clarendon Press, Oxford, 1998.

[20] David Poole. Learning, Bayesian probability, graphical models, and abduction. In P. Flach and A. Kakas, editors, *Abduction and Induction: essays on their relation and integration.* Kluwer, 1998.

[21] Helmut Prendinger and Mitsuru Ishizuka. First-order diagnosis by propositional reasoning: A representation-based approach. In *10th International Workshop on Principles of Diagnosis (DX-99)*, pages 220–225, 1999.

[22] 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.

[23] 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.

[24] 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.

[25] Gerhard Schurz. Relevance in deductive reasoning: A critical overview. In G. Schurz and M. Ursic, editors, *Beyond Classical Logic.* Academia Press, St. Augustin, 1999.

[26] 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.

[27] Markus Stumptner and Franz Wotawa. Diagnosing tree structured systems. In *Proceedings 15th International Conference on Artificial Intelligence (IJCAI-97)*, pages 440–445, 1997.

[28] Hisao Tamaki and Taisuke Sato. Unfold/fold transformation of logic programs. In *Proceedings 2nd International Logic Programming Conference*, pages 127–138, 1984.

[29] Brian C. Williams and P. Pandurang Nayak. A reactive planner for a model-based executive. In *Proceedings 15th International Conference on Artificial Intelligence (IJCAI-97)*, pages 1178–1185, 1997.