ICOT Technical Report: TR-700

TR-700

A Forward-Chaining Hypothetical Reasoner Based on Upside-Down Meta-Interpretation

by

Y. Ohta & K. Inoue

October, 1991

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Forward-Chaining Hypothetical Reasoner Based on Upside-Down Meta-Interpretation

Yoshihiko Ohta and Katsumi Inoue
ICOT Research Center
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
Phone:+81-3-3456-2514
{ohta, inoue}@icot.or.jp

Abstract

A forward-chaining hypothetical reasoner with the assumption-based truth maintenance system (ATMS) has some advantages such as avoiding repeated proofs. However, it may prove subgoals unrelated to proofs of the given goal. To simulate top-down reasoning on bottom-up reasoners, we can apply the upside-down meta-interpretation method to hypothetical reasoning. Unfortunately, when programs include negative clauses, it does not achieve speedups because checking consistency of solutions by negative clauses should be globally evaluated.

We present a new transformation algorithm of programs for efficient forward-chaining hypothetical reasoning based on the upside-down meta-interpretation. This transformation algorithm analyzes dependencies among predicate symbols in programs by using an ATMS. By this pre-dependency-analysis, we are able to restrict consistency checking to those negative clauses relevant to the given goal. The transformation algorithm has been evaluated with a logic circuit design problem.

Topic: Applications
Subtopic: Knowledge-based systems
Keywords: Hypothetical reasoning, Default reasoning, ATMS, Upside-down meta-interpretation, Program transformation

1 Introduction

Hypothetical reasoning is a technique for proving the given goal from axioms together with a set of hypotheses that do not contradict with the axioms. Hypothetical reasoning is related to abductive reasoning and default reasoning [17].

A forward-chaining hypothetical reasoner can be constructed by simply combining a bottom-up reasoner and the assumption-based truth maintenance system (ATMS) [4] (for example [7, 12]). We have implemented a forward-chaining hypothetical reasoner [11], called APRICOT/0, which consists of the RETE-based inference engine [9] and the ATMS. With this architecture, we can reduce the total cost of the label computations of the ATMS by giving intermediate justifications to the ATMS at two-input nodes in the RETE-like networks [16]. On the other hand, Poole [18] has proposed hypothetical reasoning based on top-down reasoning. Compared with top-down (backward-chaining) hypothetical reasoning, bottom-up (forward-chaining) hypothetical reasoning has the advantage of avoiding duplicate proofs of repeated subgoals and duplicate proofs among different contexts. Bottom-up reasoning, however, has the disadvantage of proving unnecessary subgoals that are unrelated to the proofs of the goal.

To avoid the disadvantage of bottom-up reasoning, both the magic set method [1, 2] and the upside-down meta-interpretation [3] have been proposed for deductive database systems. Stickel [21] has extended the upside-down meta-interpretation to abduction and deduction with non-Horn clauses. His abduction, yet, does not require consistency of solutions.

Since the consistency requirement is crucial for some applications, we would like to make programs include negative clauses for our hypothetical reasoning. When programs include negative clauses, however, the upside-down meta-interpretation method does not achieve speedups because checking consistency of solutions by negative clauses should be globally evaluated.

We present a new transformation algorithm of programs for efficient forward-chaining hypothetical reasoning based on the upside-down meta-interpretation. This transformation algorithm analyzes dependencies among predicate symbols in programs by using an ATMS. By this pre-dependency-analysis, we are able to restrict consistency checking to those negative clauses relevant to the given goal. The transformation algorithm has been evaluated with a logic circuit design problem.

In Section 2, our hypothetical reasoning is defined with the Reiter's top-down normal default proofs [19]. In Section 3, the outline of the ATMS is sketched. Section 4 shows the basic algorithm for hypothetical reasoning based on the bottom-up reasoner MGTP [10], which has been implemented in KL1 [22], with the ATMS. Section 5 presents two transformation algorithms based on the upside-down meta-interpretation. One is a simple transformation algorithm, the other is the transformation algorithm with the pre-dependency-analysis. We have implemented the hypothetical reasoner and these program transformation systems, and Section 6 shows the result of an experiment to measure run times and transformation times. In Section 7, the related works are considered.

2 Problem Definition

In this section, we define our hypothetical reasoning. A *normal default theory* (D, W) [19] and a goal g are given as follows.

- W: a set of Horn clauses.

  A *Horn clause* is represented in an implicational form,

  $$\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta \tag{1}$$

  or

  $$\alpha_1 \wedge \cdots \wedge \alpha_n \to \bot. \tag{2}$$

  Here, α_i $(0 \leq i \leq n)$ and β are atomic formulas, and $\bot$ designates falsity. All variables in a form are assumed to be universally quantified in front of the form. All variables in the consequent β have to appear in the antecedent $\alpha_1 \wedge \cdots \wedge \alpha_n$. The Horn clause that satisfies this condition is *range-restricted* [2]. A Horn clause of the form (2) is a *negative clause*.

- D: a set of normal defaults.

  A *normal default* is an inference rule,

  $$\frac{\alpha : \beta}{\beta}, \tag{3}$$

  where α, called the *prerequisite* of the normal default, is restricted to a conjunction $\alpha_1 \wedge \cdots \wedge \alpha_n$ of atomic formulas and β, called its *consequent*, is restricted to an atomic formula. All variables in the consequent β have to appear in the prerequisite α. The normal default can be read as " if α and it is consistent to assume β, then infer β".

- goal g: an atomic formula. All variables in g are assumed to be existentially quantified.

Let Δ be the set of all ground instances of the normal defaults of D. A *default proof* [19] of g with respect to (D, W) is a sequence $\Delta_0, \cdots, \Delta_k$ of subsets of Δ if and only if

1. $W \cup$ *CONSEQUENTS*$(\Delta_0) \vdash g$,
2. for $1 \leq i \leq k$,
   $W \cup$*CONSEQUENTS*$(\Delta_i) \vdash$*PREREQUISITES*(Δ_{i-1}),
3. $\Delta_k = \emptyset$,
4. $W \cup \bigcup_{i=0}^{k}$*CONSEQUENTS*(Δ_i) is consistent,

where *PREREQUISITES*$(\Delta) \equiv \bigwedge \alpha$ such that $(\alpha : \beta/\beta) \in \Delta$ and *CONSEQUENTS*$(\Delta) \equiv \{\beta \mid (\alpha : \beta/\beta) \in \Delta\}$. If $W \cup \bigcup_{i=0}^{k}$*CONSEQUENTS*$(\Delta_i) \models g\theta$, where the sequence $\Delta_0, \cdots, \Delta_k$ is a default proof of g and θ is some substitution, then θ is an answer substitution.

The task of our hypothetical reasoning is defined to find every pair of $\langle g\theta, MS(g\theta)\rangle$, where θ is an answer substitution of g and $MS(g\theta)$ is the set of minimal elements of

$$\{\ x \mid x = \bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i),\ \text{a sequence } \Delta_0, \cdots, \Delta_k \text{ is a default proof of } g\theta\ \}.$$

We call $MS(g\theta)$ the *minimal support* of $g\theta$ with respect to (D, W). As we will see later, $MS(g\theta)$ can be given as the label of $g\theta$ by the ATMS.

3 ATMS

The ATMS [4] is used as one component of our hypothetical reasoning. The outline of the ATMS is as follows.

A ground atomic formula is called a *datum*. The ATMS represents data as *ATMS nodes*:

$$\langle datum, label, justifications\rangle.$$

The ATMS treats $\perp$ and Γ_{datum}, which denotes an *assumption* of the *datum*, as special data. *Justifications* are incrementally input to the ATMS. Each justification corresponds to a ground Horn clause, and is denoted by:

$$A_1, \cdots, A_n \Rightarrow C$$

or

$$A_1, \cdots, A_n \Rightarrow \perp.$$

Here, $A_i (0 \leq i \leq n)$, called antecedents, and C, called a consequent, are ground atomic formulas. The ATMS records the set of antecedents of the justifications whose consequents correspond to the datum in the slot $justifications$ for propagating changes of labels.

Let H be a current set of assumptions. An assumption set E is called an *environment*, where $E \subseteq H$. An assumption set E is called *nogood* if $J \cup E$ derives $\perp$, where J is a current set of justifications. A *label* of a datum C is a set of environments $\{E_i\}$ that satisfy the following four properties [4, 8]:

1. the datum C holds in each environment E_i (soundness),
2. every environment in which C holds is a superset of some environment E_i (completeness),
3. each environment E_i is not nogood (consistency),
4. no environment is a subset of any other (minimality).

A basic algorithm to compute labels is shown in [4]. If the label of a datum is not empty, the datum is *believed*; otherwise it is not believed.

4 Hypothetical Reasoning with ATMS and MGTP

The MGTP prover [10] is a model generation theorem prover for checking the unsatisfiability of a first-order theory P.

Each clause in P is denoted by:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta_1 \vee \cdots \vee \beta_m,$$

where both $\alpha_i (0 \leq i \leq n)$ and $\beta_j (0 \leq j \leq m)$ are atomic formulas and all variables in β_j have to appear in α_i. The MGTP prover consists of a translator and an interpreter. The translator generates every KL1 [22] clause that corresponds to each clause in P. The range-restricted condition allows us to represent object-level variables with KL1 variables. The interpreter generates models on the basis of a set of KL1 clauses. The MGTP prover works as a bottom-up reasoner on the distributed-memory multiprocessor called Multi-PSI [14]. A first-order theory P, which is input to the MGTP prover, is called a program.

We are able to construct a hypothetical reasoner by combining MGTP as a bottom-up reasoner with the ATMS as shown in Figure 1. We can reduce the number of justifications input to the ATMS by using IN strategy [8] with MGTP, and avoid duplicate computations for determining all possible contexts of all data and maintaining consistency of multiple contexts with the ATMS. Note that the range-restricted condition allows us to use the ATMS and MGTP. The normal default theory (D, W) is translated into a program P, where **assume** is a metapredicate and

$$P = \{\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \textbf{assume}(\beta) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n : \beta / \beta) \in D\} \cup W.$$

Suppose that there is no 1-ary predicate symbol **assume** in D and W.

If there is a $(\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \mathbf{assume}(\beta))\sigma$ in a set of justifications generated by MGTP, then the ATMS [5] interprets it as $H := H \cup \{\Gamma_{\beta\sigma}\}$ and $J := J \cup \{(\alpha_1\sigma, \cdots, \alpha_n\sigma, \Gamma_{\beta\sigma} \Rightarrow \beta\sigma)\}$, where J is a current set of justifications and H is a current set of assumptions in the ATMS.

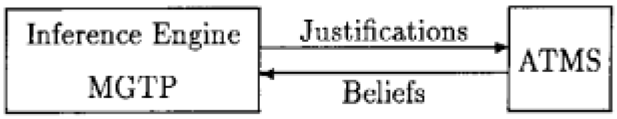

Figure 1: **Forward-Chaining Hypothetical Reasoner with ATMS and MGTP**

```
procedure  $R(g, P)$ :
begin
  $B_0 = \emptyset$;
  $i := 0$;
  repeat
    begin
      $J_{i+1} := GenerateJustifications(B_i, P)$;
      $B_{i+1} := UpdateLabels(J_{i+1}, ATMS)$;
      $i := i + 1$
    end
  until  $(B_i - B_{i-1}) = \emptyset$;
  $Solution := \emptyset$;
  for each $\theta$ such that $g\theta \in B_i$ do
    begin
      $L_{g\theta} := GetLabel(g\theta, ATMS)$;
      $Solution := Solution \cup \{(g\theta, L_{g\theta})\}$
    end;
  return  $Solution$
end.
```

Figure 2: **Reasoning Algorithm with ATMS and MGTP**

A reasoning procedure for MGTP with the ATMS, $R(g, P)$, is as shown in Figure 2. The reasoning procedure consists of the part for *GenerateJustifications-UpdateLabels* cycles and the part for constructing the solution.

Let B_i be a set of all believed data for each inference step i. Here, B_0 is empty. The MGTP prover generates a set J_{i+1} of justifications by matching elements of B_i with the antecedent of every clause related to each element of $(B_i - B_{i-1})$. We denote the procedure by $GenerateJustifications(B_i, P)$, which returns a new justification set J_{i+1}.

The ATMS updates a set of ATMS nodes by computing the labels that are just related to a justification set J_i given by MGTP. The ATMS returns a set B_i of all the data whose labels are not empty after the ATMS has updated a set of ATMS nodes with J_i given by MGTP. We denote the procedure by $UpdateLabels(J_i, ATMS)$, which returns a believed data set B_i.

The *GenerateJustifications-UpdateLabels* cycles are repeated until $(B_i - B_{i-1})$ is empty.

The procedure $GetLabel(g\theta, ATMS)$, which returns the label of the given $g\theta$, is used in the part that constructs the solution.

The hypothetical reasoner with the ATMS and MGTP can avoid duplicate proofs among different contexts and repeated proofs of subgoals. However, there are a lot of unnecessary proofs unrelated to the proofs of the goal.

5 Upside-Down Meta-Interpretation

5.1 Simple Transformation Algorithm

Bottom-up reasoning has the disadvantage of proving unnecessarily the subgoals that are not related to proofs of the given goal. There are the magic set method [1, 2] and the upside-down meta-interpretation method [3, 21] in order to allow speedups of bottom-up reasoning by incorporating goal information.

We introduce a simple transformation of program P on the basis of the upside-down meta-interpretation in [21]. A bottom-up reasoner interprets a Horn clause $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta$ in such a way that the fact $\beta\sigma$ is derived if facts $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are present for some substitution σ. On the other hand, a top-down reasoner interprets it in such a way that goals $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are derived if a goal $\beta\sigma$ is present, and fact $\beta\sigma$ is derived if both a goal $\beta\sigma$ and facts $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are present. We transform the Horn clause $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta$ into $goal(\beta) \rightarrow goal(\alpha_i)(i = 1, \cdots, n)$ and $goal(\beta) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta$, where *goal* is a metapredicate, then a bottom-up reasoner can simulate top-down reasoning if the predicate symbol *goal* does not appear in the original program P. After some facts related to the proofs of the goal have derived with the upside-down meta-interpretation, those facts may derive contradiction with bottom-up interpretation of original program. Thus, we transform each negative clause $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \perp$ into $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \perp$ and $\rightarrow goal(\alpha_i)$ for every α_i $(1 \leq i \leq n)$. This means that every subgoals related to negative clauses is evaluated.

Suppose that all arguments of the goal g are variables. Then, only the predicate symbol in the argument of the metapredicate *goal* has enough information to simulate top-down reasoning if each element in the set of n-ary predicate symbols is not in the set of m-ary predicate symbols $(n \neq m)$. Let $\bar{\beta}$ be the predicate symbol of some atomic formula β. An algorithm $T1$ as shown in Figure 3 transforms an original program P into the program $\hat{P}$ in which the top-down information is incorporated.

The solution from $T1(\bar{g}, P)$ is always the same as the solution from P because all subgoals related to negative clauses as well as the given goal are evaluated and every label of $goal(\bar{\beta})$ for some atomic formula β is $\{\emptyset\}$.

```
procedure $T1(\bar{g}, P)$ :
begin $\hat{P} := \emptyset$;
 for each $(\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow X) \in P$ do
 begin
  if $X = \perp$ then
   begin
    $\hat{P} := \hat{P} \cup \{\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \perp\}$;
    for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{\rightarrow goal(\bar{\alpha_j})\}$
   end
  else if $X = assume(\beta)$ then
   begin
    $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow assume(\beta)\}$;
    for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \rightarrow goal(\bar{\alpha_j})\}$
   end
  else if $X = \beta$ then
   begin
    $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta\}$;
    for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \rightarrow goal(\bar{\alpha_j})\}$
   end
  end;
 $\hat{P} := \hat{P} \cup \{\rightarrow goal(\bar{g})\}$;
 return $\hat{P}$
end.
```

Figure 3: **Simple Transformation Algorithm T1**

For example, consider a program,

$$
\begin{aligned}
P_b = \{ \quad & \rightarrow penguin(a), \\
& penguin(X) \rightarrow bird(X), \\
& bird(X) \rightarrow \mathbf{assume}(fly(X)), \\
& fly(X) \wedge notfly(X) \rightarrow \perp, \\
& penguin(X) \rightarrow notfly(X) \ \}.
\end{aligned}
$$

By the simple transformation algorithm, we get

$$\begin{array}{rl} T1(fly, P_b) = \{ & goal(penguin) \rightarrow penguin(a), \\ & goal(bird) \wedge penguin(X) \rightarrow bird(X), \\ & goal(bird) \rightarrow goal(penguin), \\ & goal(fly) \wedge bird(X) \rightarrow \textbf{assume}(fly(X)), \\ & goal(fly) \rightarrow goal(bird), \\ & fly(X) \wedge notfly(X) \rightarrow \bot, \\ & \rightarrow goal(fly), \\ & \rightarrow goal(notfly), \\ & goal(notfly) \wedge penguin(X) \rightarrow notfly(X), \\ & goal(notfly) \rightarrow goal(penguin) \ \} \\ \cup \ \{ & \rightarrow goal(fly) \ \}. \end{array}$$

Next, consider the goal $bird(X)$. Then, the transformed program $T1(bird, P_b)$ is the program

$$\begin{array}{rl} T1(bird, P_b) = \{ & \cdots \\ & \cdots\} \\ \cup \ \{ & \rightarrow goal(bird) \ \}, \end{array}$$

where the last element $\rightarrow goal(fly)$ of $T1(fly, P_b)$ is only replaced with $\rightarrow goal(bird)$. Both $goal(fly)$ and $goal(notfly)$ are evaluated, even if the goal is $bird(X)$. The number of firing rules in $R(bird(X), T1(bird, P_b))$, however, is nearly equal to $R(fly(X), T1(fly, P_b))$ since $T1(bird, P_b)$ as well as $T1(fly, P_b)$ includes $goal(fly)$ and $goal(notfly)$ for the negative clause.

5.2 Transformation Algorithm with Pre-Dependency-Analysis

We introduce the following theorem for omissions of consistency checking by the negative clauses that are not related to the proofs of the goal.

[Theorem]

Set

$$\bar{D} = \{(\bar{\alpha}_1 \wedge \cdots \wedge \bar{\alpha}_n : \bar{\beta}/\bar{\beta}) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n : \beta/\beta) \in D\}$$

and

$$\begin{array}{rl} \bar{W} = \{ & (\bar{\alpha}_1 \wedge \cdots \wedge \bar{\alpha}_n \rightarrow \bar{\beta}) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta) \in W\} \\ \cup \ \{ & (\bar{\alpha}_1 \wedge \cdots \wedge \bar{\alpha}_n \rightarrow false(C)) \mid C = (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot), C \in W\}. \end{array}$$

Let $\bar{S}_0 = \{\langle \bar{g}, MS(\bar{g})\rangle\}$ be the solution of $\bar{g}$ from $(\bar{D}, \bar{W})$ and $\bar{S}_c = \{\langle false(C), MS(false(C))\rangle\}$ be the solution of $false(C)$ from $(\bar{D}, \bar{W})$ for a negative clause $C \in W$.

If it holds that $E \notin MS(false(C))$ for each $E \subseteq E_{\bar{g}}$, where $E_{\bar{g}} \in MS(\bar{g})$, then the solution of g from (D, W) is equivalent to the solution of g from $(D, W - \{C\})$.

[Proof]

Let $C = (\alpha \rightarrow \bot)$ be a negative clause in W, where α is a conjunction of atomic formulas. Set the solution of g from (D, W) as

$$S_0 = \{\langle g\theta_1, P_0(g\theta_1)\rangle, \cdots, \langle g\theta_N, P_0(g\theta_N)\rangle\}$$

and the solution of g from $(D, W - \{C\})$ as

$$S_1 = \{\langle g\theta_1, P_1(g\theta_1)\rangle, \cdots, \langle g\theta_M, P_1(g\theta_M)\rangle\},$$

where θ_i $(0 \le i \le N)$ is an answer substitution of g from (D, W), θ_i $(0 \le i \le M)$ is an answer substitution of g from $(D, W - \{C\})$, $P_0(g\theta_i)$ $(0 \le i \le N)$ is the minimal support with respect to (D, W), and $P_1(g\theta_i)$ $(0 \le i \le M)$ is the minimal support with respect to $(D, W - \{C\})$. Because C is a negative clause, it is easy to see $N \le M$. And for some θ_i $(0 \le i \le M)$, it may be the case that $P_0(g\theta_i)$ and $P_1(g\theta_i)$ are different. To take these differences into account, let $S'_1 = S_0 \cup S_1$ and M' be the number of elements of S'_1. Note that $N \le M \le M'$.

Assume that $M' \ne N$. Then, there is at least one $\langle g\theta_m, P_1(g\theta_m)\rangle \in S'_1$ $(1 \le m \le M')$ such that $\langle g\theta_m, P_1(g\theta_m)\rangle \notin S_0$. Let the solution of α from $(D, W - \{C\})$ be

$$S_2 = \{\langle \alpha\sigma_1, P_1(\alpha\sigma_1)\rangle, \cdots, \langle \alpha\sigma_K, P_1(\alpha\sigma_K)\rangle\},$$

where σ_k $(0 \le k \le K)$ is an answer substitution. From the assumption $M' \ne N$, for some $E_{g\theta_m} \in P_1(g\theta_m)$, there exists $E \subseteq E_{g\theta_m}$ such that $E \in P_1(\alpha\sigma_k)$ for some σ_k $(0 \le k \le K)$.

From the condition that $E \notin MS(false(C))$ holds for each $E \subseteq E_{\bar{g}}$ and $E_{\bar{g}} \in MS(\bar{g})$, for each $E_{\bar{C}} \in MS(false(C))$, there is a predicate symbol $\bar{\pi}$ such that $\Gamma_{\bar{\pi}} \in E_{\bar{C}}$ and $\Gamma_{\bar{\pi}} \notin E_{\bar{g}}$. From the label completeness, for each σ_k $(0 \le k \le K)$, it holds that for every $E_{\alpha\sigma_k} \in P_1(\alpha\sigma_k)$ and $E_{g\theta_m} \in P_1(g\theta_m)$, there is a ground instance $\pi\rho$ of the atomic formula whose predicate symbol is $\bar{\pi}$ such that $\Gamma_{\pi\rho} \in E_{\alpha\sigma_k}$ and $\Gamma_{\pi\rho} \notin E_{g\theta_m}$. Therefore, for each $E_{g\theta_m} \in P_1(g\theta_m)$, every $E \subseteq E_{g\theta_m}$ satisfies that $E \notin P_1(\alpha\sigma_k)$ for any σ_k $(0 \le k \le K)$.

This contradicts with the assumption. Thus, if $\langle g\theta_m, P_1(g\theta_m)\rangle \in S'_1$, then $\langle g\theta_m, P_1(g\theta_m)\rangle \in S_0$. As a consequence, we have $S_0 = S_1$. **[end of proof]**

On the basis of the theorem, we can omit the consistency checking for C if the condition of the theorem is satisfied. We can obtain both $MS(false(C))$ and $E_{\bar{g}}$ by the label computations from the justification set $\bar{J}$ based on $(\bar{D}, \bar{W})$ by using an ATMS. The ATMS used in the algorithm can be seen as a dependency analyzer among the predicate symbols in the program only. So, the label computation from $\bar{J}$ is called the *pre-dependency-analysis*. This ATMS computation is not so heavy because the number of data is restricted to the number of predicate symbols and the number of assumptions is restricted to the number of assumable predicates.

```
procedure $T2(\bar{g}, P)$ :
begin
  $\hat{P} := \emptyset$;
  $\bar{J} := \emptyset$;
  $k := 1$;
  for each $(\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow X) \in P$ do
  begin
    if $X = \perp$ then
      begin
        $\hat{P} := \hat{P} \cup \{\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \perp\}$;
        $\bar{J} := \bar{J} \cup \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow false(k))\}$;
        $k := k + 1$
      end
    else if $X = assume(\beta)$ then
        begin
          $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow assume(\beta)\}$;
          $\bar{J} := \bar{J} \cup \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow assume(\bar{\beta}))\}$;
          for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \rightarrow goal(\bar{\alpha}_j)\}$
        end
      else if $X = \beta$ then
          begin
          $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta\}$;
          $\bar{J} := \bar{J} \cup \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow \bar{\beta})\}$;
          for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{goal(\bar{\beta}) \rightarrow goal(\bar{\alpha}_j)\}$
          end
  end;
  $Updatelabels(\bar{J}, ATMS)$;
  $L_g := GetLabel(\bar{g}, ATMS)$;
  for $i := 1$ until $k$ do
    begin
      $L_i := GetLabel(false(i), ATMS)$;
      for each $E_g \in L_g$ do
        for each $E_i \in L_i$ do
          if $E_i \subseteq E_g$ then
            for each $(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow false(i)) \in \bar{J}$ do
              for $j := 1$ until $n$ do $\hat{P} := \hat{P} \cup \{\rightarrow goal(\bar{\alpha}_j)\}$
    end;
  $\hat{P} := \hat{P} \cup \{\rightarrow goal(\bar{g})\}$;
  return $\hat{P}$
end.
```

Figure 4: **Transformation Algorithm T2 Using ATMS**

Figure 4 shows the transformation algorithm with the pre-dependency-analysis. The procedure transforms an original program into the program in which the top-down information is incorporated so that consistency checking is restricted to those negative clauses relevant to the given goal.

Consider the same example P_b, shown in the previous subsection, in case the goal is $bird(X)$. The justification set $\bar{J}_b$ is given to the ATMS, where

$$\begin{aligned}\bar{J}_b = \{ \quad & \Rightarrow penguin, \\ & penguin \Rightarrow bird, \\ & bird \Rightarrow \mathbf{assume}(fly), \\ & fly \wedge notfly \Rightarrow false(1), \\ & penguin \Rightarrow notfly \ \}.\end{aligned}$$

As the result of the pre-dependency-analysis, the label of $false(1)$ becomes $\{\{\Gamma_{fly}\}\}$ and the label of $bird$ becomes $\{\emptyset\}$. This means that every nogood environment associated with this negative clause becomes a superset of $\{\Gamma_{fly(X)\sigma}\}$, where σ is some ground instantiation of $fly(X)$. If the goal is $bird(X)$, then we do not need to evaluate this negative clause. The transformed program

$$\begin{aligned}T2(bird, P_b) = \{ \quad & goal(penguin) \rightarrow penguin(a), \\ & goal(bird) \wedge penguin(X) \rightarrow bird(X), \\ & goal(bird) \rightarrow goal(penguin), \\ & goal(fly) \wedge bird(X) \rightarrow \mathbf{assume}(fly(X)), \\ & goal(fly) \rightarrow goal(bird), \\ & fly(X) \wedge notfly(X) \rightarrow \bot, \\ & goal(notfly) \wedge penguin(X) \rightarrow notfly(X), \\ & goal(notfly) \rightarrow goal(penguin) \ \} \\ \cup \ \{ \quad & \rightarrow goal(bird) \ \}\end{aligned}$$

does not include $\rightarrow goal(fly)$ and $\rightarrow goal(notfly)$.

6 Evaluation with a Logic Design Problem

We have taken up the design of logic circuits to calculate the greatest common divisor (GCD) of two integers expressed in 8 bits by using the Euclidean algorithm. The solutions are circuits calculating GCD and satisfying given constraints on area and time [13]. The program [16] contains several kinds of knowledge: datapath design, component design, technology mapping, CMOS standard cells and constraints on area and time. The design problems of GCD calculators includes designing components such as subtracters and adders.

Table 1 shows the expermental result, on a Pseudo-Multi-PSI system, for the evaluation of the transformation algorithms and transformed programs. The run time of the original program

P_d is denoted by $T_{R(g,P_d)}$. The predicate symbol $\bar{g}$ of each goal g is *adder* (design of adders), *subtracter* (design of subtracters) or *cGCD* (design of GCD calculators). Each run time on the original program is nearly equal to the others because the information of the goal is used after the $GenerateJustifications - UpdateLabels$ cycles.

On the simple transformation algorithm, $T_{T1(\bar{g},P_d)}$ is the transformation time from the original program P_d into $T1(\bar{g}, P_d)$, and $T_{R(g,T1(\bar{g},P_d))}$ is the run time of $R(g, T1(\bar{g}, P_d))$. In this case, each run time is also nearly equal to the others because constraints on area and time of the GCD calculators is represented by negative clauses. Even if we want to design adders or subtracters, the hypothetical reasoner cannot avoid designing GCD calculators. This is because the program does not know whether the design of only adders or subtracters will not derive contradictions.

In the transformation algorithm with the pre-dependency-analysis, $T_{T2(\bar{g},P_d)}$ is the transformation time from the original program P_d into $T2(\bar{g}, P_d)$, and $T_{R(g,T2(\bar{g},P_d))}$ is the run time of $R(g, T2(\bar{g}, P_d))$. Each transformation time $T_{T2(\bar{g},P_d)}$ is a little bit longer (about 1 second) than each simple transformation time $T_{T1(\bar{g},P_d)}$. On the other hand, the run time of $R(g, T2(\bar{g}, P_d))$ for each design of adders and subtracters is much shorter than the run time for the design of GCD calculators. This is because the program avoids consistency checks of negative clauses representing constraints on area and time of the GCD calculators when the design of adders or subtracters is given as a goal. The results show that the total of the transformation time and the run time on the transformed program is shorter than the run time on the original program when the problem does not need the whole of the program.

Table 1: **Transformation Time and Run Time**

Goal $\bar{g}$	$T_{R(g,P_d)}$ [s]	$T_{T1(\bar{g},P_d)}$ [s]	$T_{R(g,T1(\bar{g},P_d))}$ [s]	$T_{T2(\bar{g},P_d)}$ [s]	$T_{R(g,T2(\bar{g},P_d))}$ [s]
adder	10.33	6.30	17.40	7.14	0.39
subtracter	10.30	6.28	17.15	7.05	1.72
cGCD	10.31	6.28	17.11	7.04	17.16

7 Related Work

The algorithm for first-order Horn-clause abduction with the ATMS is presented in [15]. The system is basically a consumer architecture [6] introducing backward-chaining consumers. The algorithm avoids both redundant proofs by introducing the goal-directed backward-chaining consumers and duplicate proofs among different contexts by using the ATMS.

Their problem definition is the same as [20], whose inputs are a goal and a set of Horn clauses without negative clauses. When there are negative clauses in the program, they briefly suggest that forward-chaining consumer can be used for each negative clause to check consistency. On the other hand, since we only simulate backward-chaining by the forward-chaining, we do not require both types of chaining rules. Moreover, we see that when the program includes negative clauses, it is sometimes difficult to represent the clauses as a set of consumers.

For example, suppose that the axioms are

$$\{a \rightarrow c,\ b \rightarrow d,\ c \wedge d \rightarrow g,\ c \rightarrow e,\ d \rightarrow f,\ e \wedge f \rightarrow \perp\}$$

and a goal is g. Assume a set of consumers

$$\{(c \Leftarrow a),\ (d \Leftarrow b),\ (g \Leftarrow c, d),\ (e \Leftarrow c),\ (f \Leftarrow d),\ (e, f \Rightarrow \perp)\},$$

where "$\Leftarrow$" means a backward-chaining consumer and "$\Rightarrow$" means a forward-chaining consumer. Then, we get the solution $\{\langle g, \{\{\Gamma_g\}, \{\Gamma_a, \Gamma_b\}, \{\Gamma_a, \Gamma_d\}, \{\Gamma_c, \Gamma_b\}, \{\Gamma_c, \Gamma_d\}\}\rangle\}$.

However, the correct solution is $\{\langle g, \{\{\Gamma_g\}\}\rangle\}$ because $\{\Gamma_a, \Gamma_b\}, \{\Gamma_a, \Gamma_d\}, \{\Gamma_c, \Gamma_b\}$ and $\{\Gamma_c, \Gamma_d\}$ are nogood. To guarantee the consistency when the program includes negative clauses, for every Horn clause, we have to add the corresponding forward-chaining consumer. Such added consumers would cause the same problem as the programs obtained by using the simple transformation algorithm.

In [21], deduction and abduction with the upside-down meta-interpretation are proposed. This abduction does not require consistency of solutions. Furthermore, rules may do duplicate firing in different contexts since it does not use the ATMS. This often causes a problem when it is applied to practical programs where heavy procedures are attached to rules.

The other difference between the frameworks of [15, 21] and ours is that the frameworks of [15, 21] treat only hypotheses in the form of normal defaults without prerequisites, whereas we allow for normal defaults with prerequisites in theories.

8 Conclusion

We have presented a new transformation algorithm of programs for efficient forward-chaining hypothetical reasoning based on the upside-down meta-interpretation. An ATMS is used in the algorithm in order to analyze dependencies among predicate symbols in programs in order to restrict consistency checking only to those negative clauses relevant to the given goal. It has been evaluated by using a logic circuit design problem on a Pseudo-Multi-PSI system.

Acknowledgments

Thanks are due to Mr. Makoto Nakashima of JIPDEC for implementing the ATMS and combining it with MGTP. We would like to express our appreciation to Professor Mitsuru Ishizuka of the University of Tokyo and the other members of ICOT ADS-Working Group for the helpful discussion. We would like to thank Dr. Ryuzo Hasegawa and Mr. Miyuki Koshimura for giving us the MGTP prover and for their support. We would also like to thank Dr. Koichi Furukawa for his advise. Finally, we would like to express our appreciation to Dr. Kazuhiro Fuchi, Director of the Institute for New Generation Computer Technology, who provided us with the opportunity to conduct this research.

References

[1] Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J. D., Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. of the ACM PODS*, pp.1-15 (1986).

[2] Bancilhon, F. and Ramakrishnan, R., Performance Evaluation of Data Intensive Logic Programs, Minker, J.(Ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, pp.439–516 (1987).

[3] Bry, F.,Query evaluation in recursive databases: bottom-up and top-down reconciled, *Data & Knowledge Engineering*, **5**, pp.289–312 (1990).

[4] de Kleer, J., An Assumption-based TMS, *Artificial Intelligence*, **28**, pp.127–162 (1986).

[5] de Kleer, J., Extending the ATMS, *Artificial Intelligence*, **28**, pp.163–196 (1986).

[6] de Kleer, J., Problem Solving with the ATMS, *Artificial Intelligence*, **28**, pp.197–224 (1986)

[7] Flann, N. S., Dietterich, T. G. and Corpron, D. R., Forward Chaining Logic Programming with the ATMS, *Proceedings of AAAI-87*, pp.24-29 (1987).

[8] Forbus, K. D. and de Kleer, J., Focusing the ATMS, *Proc. of 7th National Conference on Artificial Intelligence*, pp.193–198 (1988).

[9] Forgy, C. L., Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, **19**, pp.17-37 (1982).

[10] Fujita, H. and Hasegawa, R., A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm, *Proc. of the 8th International Conference on Logic Programming*, pp.494–500 (1991).

[11] Inoue, K., Problem Solving with Hypothetical Reasoning, *Proc. of the International Conference on Fifth Generation Computer Systems*, **3**, pp.1275-1281 (1988).

[12] Junker, U., *Reasoning in Multiple Contexts*, GMD Working Paper No.334 (1988).

[13] Maruyama, F., Kakuda, T., Masunaga, Y., Minoda, Y., Sawada, S. and Kawato, N., co-LODEX: A Cooperative Expert System for Logic Design, *Proc. of the International Conference on Fifth Generation Computer Systems*, **3**, pp.1299-1306 (1988).

[14] Nakajima, K., Inamura, Y., Ichiyoshi, N., Rokusawa, K. and Chikayama, T., Distributed Implementation of KL1 on the Multi-PSI/V2, *Proc. of the 6th International Conference on Logic Programming*, pp.436–451 (1989).

[15] Ng, H. T. and Mooney, R., J., *An Efficient First-Order Abduction System Based on the ATMS*, Technical Report AI 91-151, The University of Texas at Austin, AI Lab. (1991).

[16] Ohta, Y. and Inoue, K., A Forward-Chaining Multiple-Context Reasoner and Its Application to Logic Design, *Proc. of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pp.386–392 (1990).

[17] Poole, D., A Logical Framework for Default Reasoning, *Artificial Intelligence*, **36**, pp.27–47 (1988).

[18] Poole, D., Compiling a Default Reasoning System into Prolog, *New Generation Computing*, **9**, pp.3–38 (1991).

[19] Reiter, R., A Logic for Default Reasoning, *Artificial Intelligence*, **13**, pp.81–132 (1980).

[20] Stickel, M., E., Rationale and Methods for Abductive Reasoning in Natural-Language Interpretation, *Lecture Nodes in Artificial Intelligence* # 459, Springer-Verlag, pp.233–252 (1990).

[21] Stickel, M., E., *Upside-Down Meta-Interpretation of the Model Elimination Theorem-Prover Procedure for Deduction and Abduction*, ICOT Technical Report TR-664, ICOT Research Center (1991).

[22] Ueda, K. and Chikayama, T., Design of the Kernel Language for the Parallel Inference Machine, *The Computer Journal*, **33**, No.6, pp. 494–500 (1990).