PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

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

Yoshihiko Ohta Katsumi Inoue
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F, 1-4-28 Mita, Minato-ku, Tokyo 108, Japan
{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 the consistency of solutions by negative clauses should be globally evaluated. This paper describes a new transformation algorithm of programs for efficient forward-chaining hypothetical reasoning. In the transformation algorithm, logical dependencies between a goal and negative clauses are analyzed to find irrelevant negative clauses, so that the forward-chaining hypothetical reasoners based on the upside-down meta-interpretation can restrict consistency checking of negative clauses to those relevant clauses. The transformed program has been evaluated with a logic circuit design problem.

1 Introduction

Hypothetical reasoning [Inoue 88] 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.

A forward-chaining hypothetical reasoner can be constructed by simply combining a bottom-up reasoner with the assumption-based truth maintenance system (ATMS) [de Kleer 86-1] (for example [Flann *et al.* 87, Junker 88]). We have implemented a forward-chaining hypothetical reasoner [Ohta and Inoue 90], called APRICOT/0, which consists of the RETE-based inference engine [Forgy 82] 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. On the other hand, hypothetical reasoning based on top-down reasoning has been proposed in [Poole *et al.* 87, Poole 91]. 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, Magic Set method [Bancilhon *et al.* 86] and Alexander method [Rohmer *et al.* 86] have been proposed for deductive database systems. Recently, it is shown that Magic Set and Alexander methods are interpreted as specializations of the upside-down meta-interpretation [Bry 90]. The upside-down meta-interpretation has been extended to abduction and deduction with non-Horn clauses in [Stickel 91]. His abduction, however, does not require the 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 the 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. In the transformation algorithm, logical dependencies between a goal and negative clauses are analyzed to find irrelevant negative clauses, so that the forward-chaining hypothetical reasoners based on the upside-down meta-interpretation can restrict consistency checking of negative clauses to those relevant clauses. The transformed program has been evaluated with a logic circuit design problem.

In Section 2, our hypothetical reasoning is defined with the default proofs [Reiter 80]. 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 [Fujita and Hasegawa 91] together 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 abstracted dependency analysis. We have implemented the hypothetical reasoner and these program transformation systems, and Section 6 shows the result of an experiment for the evaluation of the transformed programs. In Section 7, related works are considered.

2 Problem Definition

In this section, we define our hypothetical reasoning based on a subset of normal default theories [Reiter 80]. A normal default theory (D, W) and a goal G are given as follows:

- W: a set of Horn clauses.

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

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

  or

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

  Here, α_i $(1 \leq i \leq n; n \geq 0)$ and β are atomic formulas, and $\bot$ designates falsity. Function symbols are restricted to 0-ary function symbols. All variables in a clause are assumed to be universally quantified in front of the clause. Each Horn clause has to be *range-restricted*, that is, all variables in the consequent β have to appear in the antecedent $\alpha_1 \wedge \cdots \wedge \alpha_n$. A Horn clause of the form (2) is called 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. Function symbols are restricted to 0-ary function symbols. All variables in the consequent β have to appear in the prerequisite α. A normal default with free variables is identified with the set of its ground instances. The normal default can be read as " if α and it is consistent to assume β, then infer β".

- goal G: a conjunction of atomic formulas.

  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* [Reiter 80] 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 $1 \leq i \leq k$,
   $W \cup CONSEQUENTS(\Delta_i) \vdash PREREQUISITES(\Delta_{i-1})$,
3. $\Delta_k = \emptyset$,
4. $W \cup \bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i)$ is consistent,

where

$$PREREQUISITES(\Delta_{i-1}) \equiv \bigwedge \alpha$$

for $(\alpha : \beta/\beta) \in \Delta_{i-1}$ and

$$CONSEQUENTS(\Delta_i) \equiv \{\beta \mid (\alpha : \beta/\beta) \in \Delta_i\}.$$

A ground instance $G\theta$ of the goal G is an *answer* to G from (D, W) 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 with respect to (D, W). If $G\theta$ is an answer to G from (D, W), θ is an *answer substitution* for G from (D, W). A *support* for an answer $G\theta$ from (D, W) is $\bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i)$, where the sequence $\Delta_0, \cdots, \Delta_k$ is a default proof of $G\theta$ with respect to (D, W). For an answer $G\theta$ from (D, W), the *minimal supports* for $G\theta$ from (D, W), written as $MS(G\theta)$, is the set of minimal elements in all supports for $G\theta$ from (D, W). The *solution* to G from (D, W) is the set of all pairs $\langle G\theta, MS(G\theta)\rangle$, where $G\theta$ is an answer to G from (D, W) and $MS(G\theta)$ is the minimal supports for $G\theta$. The task of our hypothetical reasoning is defined to find the solution to a given goal from a given normal default theory.

3 ATMS

The ATMS [de Kleer 86-1] is used as one component of our hypothetical reasoner. The following is the outline of the ATMS.

In the ATMS, a ground atomic formula is called a *datum*. For some datum N, Γ_N designates an *assumption*. The ATMS treats both $\bot$ and Γ_N as special data. The ATMS represents each datum as an *ATMS node*:

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

Justifications correspond to ground Horn clauses and are incrementally input to the ATMS. Each justification is denoted by:

$$N_1, \cdots, N_n \Rightarrow N,$$

where N_i and N are data. Each datum N_i is called an antecedent, and the datum N is called a consequent. In the slot *justifications*, the ATMS records the set of antecedents of justifications whose consequents correspond to the *datum*.

Let H be a current set of assumptions. An assumption set $E \subseteq H$ is called an *environment*. When we denote an environment by a set of assumptions, each assumption Γ_N is written as N by omitting the letter Γ. Let J be a current set of justifications. An environment E is called *nogood* if $J \cup E$ derives $\bot$. The *label* of the datum N is the set of environments $\{E_1, \cdots, E_j, \cdots, E_m\}$ that satisfies the following four properties [de Kleer 86-1]:

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

If the label of a datum is not empty, the datum is *believed*; otherwise it is not believed. A basic algorithm to compute labels [de Kleer 86-1] is as follows. When a justification is incrementally input to the ATMS, the ATMS updates the labels relevant to the justification in the following procedure.

Step 1: Let L be the current label of the consequent N of the justification and L_i be the current label of the i-th antecedent N_i of the justification. Set $L' = L \cup \{ x \mid x = \bigcup_{i=1}^{n} E_i, \ where \ E_i \in L_i \}$.

Step 2: Let L'' be the set obtained by removing nogoods and subsumed environments from L'. Set the new label of N to L''.

Step 3: Finish this updating if L is equal to the new label.

Step 4: If N is $\bot$, then remove all new nogoods from labels of all data other than $\bot$.

Step 5: Update labels of the consequents of the recorded justifications which contain N as their antecedents.

4 Hypothetical Reasoner with ATMS and MGTP

The MGTP [Fujita and Hasegawa 91] 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 $\alpha_i (1 \leq i \leq n; n \geq 0)$ and $\beta_j (1 \leq j \leq m; m \geq 0)$ are atomic formulas and all variables in $\beta_1 \vee \cdots \vee \beta_m$ have to appear in $\alpha_1 \wedge \cdots \wedge \alpha_n$. Each clause in P is translated into a KL1 [Ueda and Chikayama 90] clause. Then, model candidates are generated from the set of KL1 clauses. The MGTP works as a bottom-up reasoner on the distributed-memory multiprocessor called Multi-PSI.

As shown in Figure 1, we can construct a hypothetical reasoner by combining the MGTP with the ATMS. The normal default theory (D, W) is translated into a program P,

$$P \equiv \{ \ \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \mathbf{assume}(\beta) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n : \beta / \beta) \in D \ \} \cup W,$$

where **assume** is a metapredicate not appearing anywhere in D and W.

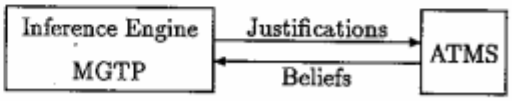

Figure 1: **Forward-Chaining Hypothetical Reasoner with ATMS and MGTP**

```
procedure R(G, P):
begin
  B_0 := ∅;
  J_0 := { (⇒ β) | (→ β) ∈ P }
       ∪ { (Γ_β ⇒ β) | (→ assume(β)) ∈ P };
  s := 0;
  while J_s ≠ ∅ do
    begin
      s := s + 1;
      B_s := UpdateLabels(J_{s-1}, ATMS);
      J_s := GenerateJustifications(B_s, P, B_{s-1})
    end;
  Solution := ∅;
  for each θ such that Gθ ∈ B_s do
    begin
      L_{Gθ} := GetLabel(Gθ, ATMS);
      Solution := Solution ∪ {(Gθ, L_{Gθ})}
    end;
  return Solution
end.
```

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

The reasoning procedure $R(G, P)$ for the MGTP with the ATMS is shown in Figure 2. The reasoning proce-

dure consists of the part for *UpdateLabels* – *GenerateJustifications* cycles and the part for constructing the solution. The *UpdateLabels* – *GenerateJustifications* cycles are repeated while J_s is not empty. The ATMS updates the labels related to a justification set J_{s-1} given by the MGTP. The ATMS returns the set B_s of all the data whose labels are not empty after the ATMS has updated labels with J_{s-1}. The procedure $UpdateLabels(J_{s-1}, ATMS)$ returns a believed data set B_s. The MGTP generates each set J_s of justifications by matching elements of B_s with the antecedent of every clause related to new believed data. The procedure $GenerateJustifications(B_s, P, B_{s-1})$ returns a new justification set J_s. If any element in $(B_s \setminus B_{s-1})$ can match an element of the antecedent of any $(\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow X)$ in P and there exists a ground substitution σ for all α_i such that $\alpha_i\sigma \in B_s$, then J_s is as follows.

- $(\alpha_1\sigma, \cdots, \alpha_n\sigma, \Gamma_{\beta\sigma} \Rightarrow \beta\sigma) \in J_s$ if $X = \mathrm{assume}(\beta)$.
- $(\alpha_1\sigma, \cdots, \alpha_n\sigma \Rightarrow \beta\sigma) \in J_s$ if $X = \beta$.
- $(\alpha_1\sigma, \cdots, \alpha_n\sigma \Rightarrow \bot) \in J_s$ if $X = \bot$.

The procedure $GetLabel(G\theta, ATMS)$ returns the label of $G\theta$ and is used in constructing the solution. Note that the label of $G\theta$ corresponds to the minimal supports for $G\theta$. The hypothetical reasoner with the ATMS and the MGTP can avoid duplicate proofs among different contexts and repeated proofs of subgoals. However, there may be 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 subgoals that are not related to proofs of the given goal. We introduce a simple transformation of a program P on the basis of the upside-down meta-interpretation for speedups of bottom-up reasoning by incorporating goal information. 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

$$goal(\beta) \rightarrow goal(\alpha_i)$$

for every α_i $(1 \leq i \leq n)$ and

$$goal(\beta) \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta,$$

then a bottom-up reasoner can simulate top-down reasoning. Here, *goal* is a metapredicate symbol which 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 the original program. Thus, we transform each negative clause

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot$$

into

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot$$

and

$$\rightarrow goal(\alpha_i)$$

for every α_i $(1 \leq i \leq n)$. This means that every subgoal related to negative clauses is evaluated.

Note that $(goal(\beta) \rightarrow goal(\alpha_i))$ or $(\rightarrow goal(\alpha_i))$ may not be satisfy the range-restricted condition. We have some techniques which make every clause in transformed programs range-restricted. Here, we take a very simple technique in which only the predicate symbols are used as the arguments of the metapredicate *goal*. When γ is an atomic formula, we denote by $\tilde{\gamma}$ the predicate symbol of γ. The 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 to G from $T1(\bar{G}, P)$ is always the same as the solution to G from P because all subgoals related to negative clauses as well as the given goal are evaluated and every label of $goal(\tilde{\beta})$ for any atomic formula β is $\{\emptyset\}$.

For example, consider a program,

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

By the simple transformation algorithm, we get

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

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

$$T1(bird, P_b) = \{\cdots\} \cup \{ \rightarrow goal(bird) \},$$

where only the last element $(\rightarrow goal(fly))$ of $T1(fly, P_b)$ is replaced with $(\rightarrow goal(bird))$. Even if the goal is $bird(X)$, both $goal(fly)$ and $goal(notfly)$ are evaluated because $\{\cdots\}$ includes $(\rightarrow goal(fly))$ and $(\rightarrow goal(notfly))$ for the negative clause. Then, the computational cost of $R(bird(X), T1(bird, P_b))$ is nearly equal to the cost of $R(fly(X), T1(fly, P_b))$.

```
procedure T1(G, P):
begin
  P̂ := ∅;
  for each (α1 ∧ ··· ∧ αn → X) ∈ P do
  begin
    if X = ⊥ then
      begin
        P̂ := P̂ ∪ {α1 ∧ ··· ∧ αn → ⊥};
        for j := 1 until n do
          P̂ := P̂ ∪ {→ goal(ᾱj)}
      end
    else if X = assume(β) then
      begin
        P̂ := P̂ ∪ {goal(β̄) ∧ α1 ∧ ··· ∧ αn → assume(β)};
        for j := 1 until n do
          P̂ := P̂ ∪ {goal(β̄) → goal(ᾱj)}
      end
    else if X = β then
      begin
        P̂ := P̂ ∪ {goal(β̄) ∧ α1 ∧ ··· ∧ αn → β};
        for j := 1 until n do
          P̂ := P̂ ∪ {goal(β̄) → goal(ᾱj)}
      end
    end;
  P̂ := P̂ ∪ {→ goal(Ḡ)};
  return P̂
end.
```

Figure 3: **Simple Transformation Algorithm** $T1$

5.2 Transformation Algorithm with Abstracted Dependency Analysis

In this subsection, we describe a static method to find irrelevant negative clauses to evaluation of the goal. If we can find such irrelevant negative clauses, for every antecedent α_i of each irrelevant clause, we do not need to add $(\rightarrow goal(\alpha_i))$ into the transformed program. We try to find them by analyzing logical dependencies between the goal and each negative clause at the abstracted level. We do not care about any argument in the abstracted dependency analysis.

When γ is an atomic formula, we denote by the proposition $\bar{\gamma}$ the predicate symbol of γ. For each negative clause C, the proposition $false(C)$ is used as the identifier of C. For every $(\alpha \rightarrow \mathbf{assume}(\beta))$, $\bar{\beta}$ is called an *assumable-predicate symbol*. For any environment E, its *abstracted environment* (denoted by $\bar{E}$) is $\{ \Gamma_{\bar{\beta}} \mid \Gamma_\beta \in E \}$. The *abstracted justifications* with respect to P is defined as:

$$\begin{aligned}\bar{J} \equiv\ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n, \Gamma_{\bar{\beta}} \Rightarrow \bar{\beta}) \mid \\ & \qquad (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \mathbf{assume}(\beta)) \in P\} \\ \cup\ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow \bar{\beta}) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta) \in P\} \\ \cup\ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow false(C)) \mid \\ & \qquad C = (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot),\ C \in P\}.\end{aligned}$$

Let $\bar{A}$ be the set of propositions appearing in $\bar{J}$. Note that $\bar{A}$ consists of all predicate symbols in P and all $false(C)$ for $C \in P$. For each proposition N in $\bar{A}$, we compute a set of abstracted environments on which N depends. Now, we show an algorithm to compute the set of abstracted environments. This algorithm is obtained by modifying the label-updating algorithm shown in Section 3. The modified points are as follows.

1. Replace **Step 2** with
   Step 2′: Set the new label of N to L'.
2. Remove **Step 4**.

Every proposition in $\bar{A}$ is labeled with the set of abstracted environments obtained by applying the modified algorithm to the abstracted justifications $\bar{J}$. This label is called the *abstracted label* of the proposition. The system to compute the set of abstracted environments for each proposition is called an *abstracted dependency analyzer*. The reasons why we have to modify the label-updating algorithm are as follows. Firstly, in the abstracted justifications, every $\bot$ is replaced with the proposition $false(C)$ for the negative clause C, so that each abstracted label is always consistent. Thus, we do not need **Step 4**. Secondly, each abstracted label may not be minimal because we replace **Step 2** with **Step 2′**. Suppose that every abstracted label is minimal. Then, the theorem we present below may not hold. For example, let

$$\begin{aligned}P_e = \{\ & \rightarrow p(a),\ \rightarrow p(b),\ \rightarrow q(b),\ q(X) \rightarrow t(X), \\ & p(X) \rightarrow \mathbf{assume}(r(X)), \\ & p(X) \rightarrow \mathbf{assume}(s(X)), \\ & r(a) \rightarrow g,\ r(X) \wedge s(X) \rightarrow g, \\ & r(X) \wedge s(X) \wedge t(X) \rightarrow \bot\ \}.\end{aligned}$$

Consider the problem defined with the goal g and P_e. The abstracted label of g is $\{\{r\}, \{r, s\}\}$. The abstracted label of the negative clause is $\{\{r, s\}\}$. The abstracted environment $\{r, s\}$ cannot be omitted for g although the set of minimal elements in the abstracted label of g is $\{\{r\}\}$.

```
procedure T2(Ḡ, P) :
begin
  P̂ := ∅;
  J̄ := ∅;
  k := 0;
  for each (α1 ∧ ··· ∧ αn → X) ∈ P do
  begin
    if X = ⊥ then
      begin
        k := k + 1
        P̂ := P̂ ∪ {α1 ∧ ··· ∧ αn → ⊥};
        J̄ := J̄ ∪ {(ᾱ1, ··· , ᾱn ⇒ false(k))};
      end
    else if X = assume(β) then
      begin
        P̂ := P̂ ∪
           {goal(β̄) ∧ α1 ∧ ··· ∧ αn → assume(β)};
        J̄ := J̄ ∪ {(ᾱ1, ··· , ᾱn, Γβ ⇒ β̄)};
        for j := 1 until n do
          P̂ := P̂ ∪ {goal(β̄) → goal(ᾱj)}
      end
    else if X = β then
      begin
        P̂ := P̂ ∪ {goal(β̄) ∧ α1 ∧ ··· ∧ αn → β};
        J̄ := J̄ ∪ {(ᾱ1, ··· , ᾱn ⇒ β̄)};
        for j := 1 until n do
          P̂ := P̂ ∪ {goal(β̄) → goal(ᾱj)}
      end
  end;
  UpdateAbstractedLabels(J̄, ADA);
  LG := GetAbstractedLabel(Ḡ, ADA);
  for i := 1 until k do
    begin
      Li := GetAbstractedLabel(false(i), ADA);
      for each EG ∈ LG do
        for each Ei ∈ Li do
          if Ei ⊆ EG then
            for (ᾱ1, ··· , ᾱn ⇒ false(i)) ∈ J̄ do
              for j := 1 until n do
                P̂ := P̂ ∪ {→ goal(ᾱj)}
    end;
  P̂ := P̂ ∪ {→ goal(Ḡ)};
  return P̂
end.
```

Figure 4: Transformation Algorithm $T2$ with Abstracted Dependency Analysis

Theorem: Let P be a normal default theory and G a goal, $\bar{J}$ the abstracted justifications with respect to P , $L(\bar{G})$ the abstracted label of $\bar{G}$, $L(false(C))$ the abstracted label of $false(C)$ where $C \in P$. If no element in $L(false(C))$ is a subset of any element in $L(\bar{G})$, then the solution to G from P is equivalent to the solution to G from $P \setminus \{C\}$.

Sketch of the proof: Let C be $(\alpha \rightarrow \bot)$ and P' be $P \setminus \{C\}$. Assume that θ_m is any answer substitution for G from P' and σ_k is any answer substitution for α from P'. Let $MS(\alpha\sigma_k)$ be the minimal supports for $\alpha\sigma_k$ from P' and $MS(G\theta_m)$ be the minimal supports for $G\theta_m$ from P'. Suppose that no element in $L(false(C))$ is a subset of any element in $L(\bar{G})$. From the supposition and similarity between ATMS labels and abstracted labels, no element in $MS(\alpha\sigma_k)$ is a subset of any element in $MS(G\theta_m)$. Therefore, the solution to G from $P' \cup \{C\}$ is the same as the solution to G from P'. ■

On the basis of the theorem, we can omit consistency checking for a negative clause C if the condition of the theorem is satisfied. The transformation algorithm $T2(\bar{G}, P)$ with the abstracted dependency analysis is shown in Figure 4 for the program P and the goal G. In Figure 4, $UpdateAbstractedLabels(\bar{J}, ADA)$ denotes the procedure which computes abstracted labels from abstracted justifications $\bar{J}$ with the abstracted dependency analyzer ADA, and $GetAbstractedLabel(\bar{G}, ADA)$ denotes the procedure which returns the abstracted label of $\bar{G}$ from the abstracted dependency analyzer ADA. The procedure transforms an original program into the program in which the top-down information is incorporated and 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 that the goal is $bird(X)$. The abstracted justifications $\bar{J}_b$ is

$$\{\ (\Rightarrow penguin), (penguin \Rightarrow bird), (bird, \Gamma_{fly} \Rightarrow fly),\ (fly, notfly \Rightarrow false(1)),\ \ (penguin \Rightarrow notfly)\ \}.$$

As the result of the abstracted dependency analysis, the abstracted label of $false(1)$ is $\{\{fly\}\}$ and the abstracted label of $bird$ is $\{\emptyset\}$. Then, no element in the abstracted label of $false(1)$ is a subset of any element in the abstracted label of $bird$, so that we do not need to evaluate this negative clause. As a consequence, we have the transformed program:

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

Since the transformed program does not include $(\rightarrow goal(fly))$ and $(\rightarrow goal(notfly))$, the reasoner can omit solving both the goal $fly(X)$ and the goal $notfly(X)$.

6 Evaluation with 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 [Maruyama *et al.* 88]. The program P_d contains several kinds of knowledge: datapath design, component design, technology mapping, CMOS standard cells and constraints on area and time [Ohta and Inoue 90]. The design problem of calculators for GCD includes design of components such as subtracters and adders.

Table 1 shows the expermental result, on a Pseudo-Multi-PSI system, for the evaluation of the transformed programs. The run time of a program P for a goal G is denoted by $T_{R(G,P)}$. The predicate symbol $\bar{G}$ of each goal G is *adder* (design of adders), *subtracter* (design of subtracters) or *cGCD* (design of calculators for GCD). The run time $T_{R(G,P_d)}$ of each goal G is equal to the others on the original program P_d.

Table 1: Run Time of Program

Goal G	$T_{R(G,P_d)}$ [s]	$T_{R(G,P_1)}$ [s]	$T_{R(G,P_2)}$ [s]
adder	10.7	17.5	0.4
subtracter	10.7	17.3	0.6
cGCD	10.7	17.3	16.8

Let P_1 be the simple transformed program of P_d. The experiment on the simple transformation time shows that it takes 6.35 [s] for making P_1 from P_d. However, the run time $T_{R(G,P_1)}$ for each goal G is nearly equal to the others because constraints on area and time of the GCD calculators are represented by negative clauses. Even if we want to design adders or subtracters, the hypothetical reasoner cannot avoid designing GCD calculators for consistency checking.

Let P_2 be the transformed program with the abstracted dependency analysis. The experiment on the transformation time with the abstracted dependency analysis shows that it takes 6.63 [s] for making P_2 from P_d. The transformation time with the abstracted dependency analysis is a little bit longer (0.28 [s]) than the simple transformation time. When $\bar{G}$ is *adder* or *subtracter*, the run time $T_{R(G,P_2)}$ is much shorter than the run time for the design of GCD calculators. This is because the program can avoid consistency checks for negative clauses representing constraints on area and time of the GCD calculators when the design of adders or the design of subtracters is given as a goal. The result show that each total of the transformation time with abstracted dependency analysis and the run time of the transformed program is shorter than the run time of the original program when the problem does not need the whole of the program.

7 Related Work

The algorithm for first-order Horn-clause abduction with the ATMS is presented in [Ng and Mooney 91]. The system is basically a consumer architecture [de Kleer 86-3] 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 [Stickel 90], 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 the consistency. On the other hand, since we only simulate backward-chaining by the forward-chaining reasoner, 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

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

and the goal is g. Assume that the set of consumers is

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

where $\Leftarrow$ means a backward-chaining consumer and $\Rightarrow$ means a forward-chaining consumer. Then, we get the solution $\{(g, \{\{g\}, \{a, b\}, \{a, d\}, \{c, b\}, \{c, d\}\})\}$. However, the correct solution is $\{(g, \{\{g\}\})\}$ because $\{a, b\}, \{a, d\}, \{c, b\}$ and $\{c, d\}$ 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 program that appeared in using the simple transformation algorithm.

In [Stickel 91], deduction and abduction with the upside-down meta-interpretation are proposed. This abduction does not require the 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.

Another difference between the frameworks of [Ng and Mooney 91, Stickel 91] and ours is that their

frameworks treat only hypotheses in the form of normal defaults without prerequisites, whereas we allow for normal defaults with prerequisites.

8 Conclusion

We have presented a new transformation algorithm of programs for efficient forward-chaining hypothetical reasoning based on the upside-down meta-interpretation. In the transformation algorithm, logical dependencies between a goal and negative clauses are analyzed at abstracted level to find irrelevant negative clauses, so that consistency checking of negative clauses can be restricted to those relevant clauses. It has been evaluated with a logic circuit design problem on a Pseudo-Multi-PSI system.

We can also apply this abstracted dependency analysis to transformed programs based on Magic Set and Alexander methods. Our dependency analysis with only predicate symbols may be extended to an analysis with predicate symbols and their some arguments.

Acknowledgments

Thanks are due to Mr. Makoto Nakashima of JIPDEC for implementing the ATMS and combining it with the MGTP. We are grateful to Prof. Mitsuru Ishizuka of the University of Tokyo for the helpful discussion. We would also like to thank Dr. Ryuzo Hasegawa and Mr. Miyuki Koshimura for providing us the MGTP, and Dr. Koichi Furukawa for his advise. Finally, we would like to express our appreciation to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who provided us with the opportunity to conduct this research.

References

[Bancilhon *et al.* 86] F. Bancilhon, D. Maier, Y. Sagiv and J.D. Ullman, Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. of ACM PODS*, pp.1-15 (1986).

[Bry 90] F. Bry, Query evaluation in recursive databases: bottom-up and top-down reconciled, *Data & Knowledge Engineering*, 5, pp.289–312 (1990).

[de Kleer 86-1] J. de Kleer, An Assumption-based TMS, *Artificial Intelligence*, 28, pp.127–162 (1986).

[de Kleer 86-2] J. de Kleer, Extending the ATMS, *Artificial Intelligence*, 28, pp.163–196 (1986).

[de Kleer 86-3] J. de Kleer, Problem Solving with the ATMS, *Artificial Intelligence*, 28, pp.197–224 (1986)

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

[Forgy 82] C.L. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, 19, pp.17-37 (1982).

[Fujita and Hasegawa 91] H. Fujita and R. Hasegawa, A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm, *Proc. of ICLP '91*, pp.494–500 (1991).

[Inoue 88] K. Inoue, Problem Solving with Hypothetical Reasoning, *Proc. of FGCS '88*, pp.1275-1281 (1988).

[Junker 88] U. Junker, Reasoning in Multiple Contexts, GMD Working Paper No.334 (1988).

[Maruyama *et al.* 88] F. Maruyama, T. Kakuda, Y. Masunaga, Y. Minoda, S. Sawada and N. Kawato, co-LODEX: A Cooperative Expert System for Logic Design, *Proc. of FGCS '88*, pp.1299-1306 (1988).

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

[Ohta and Inoue 90] Y. Ohta and K. Inoue, A Forward-Chaining Multiple-Context Reasoner and Its Application to Logic Design, *Proc. of IEEE TAI*, pp.386–392 (1990).

[Poole *et al.* 87] D. Poole, R. Goebel and R. Aleliunas, Theorist: A logical Reasoning System for Defaults and Diagnosis, N. Cercone and G. McCalla (Eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer-Verlag, pp.331–352 (1987).

[Poole 91] D. Poole, Compiling a Default Reasoning System into Prolog, *New Generation Computing*, 9, pp.3–38 (1991).

[Reiter 80] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, 13, pp.81–132 (1980).

[Rohmer *et al.* 86] J. Rohmer, R. Lescoeur and J.M. Kerisit, The Alexander Method — A Technique for The Processing of Recursive Axioms in Deductive Databases, *New Generation Computing*, 4, pp.273–285 (1986).

[Stickel 90] M.E. Stickel, Rationale and Methods for Abductive Reasoning in Natural-Language Interpretation, *Lecture Nodes in Artificial Intelligence*, 459, Springer-Verlag, pp.233–252 (1990).

[Stickel 91] M.E. Stickel, Upside-Down Meta-Interpretation of the Model Elimination Theorem-Prover Procedure for Deduction and Abduction, ICOT Technical Report TR-664, ICOT (1991).

[Ueda and Chikayama 90] K. Ueda and T. Chikayama, Design of the Kernel Language for the Parallel Inference Machine, *The Computer Journal*, 33, 6, pp. 494–500 (1990).