ICOT Technical Report: TR-625

TR-625

A Bottom-up Procedure with Top-down Expectation for General Logic Program with Integrity Constraint

by
N. Iwayama & K. Satoh

February, 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 Bottom-up Procedure with Top-down Expectation for General Logic Program with Integrity Constraint

Noboru Iwayama, Ken Satoh
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
email: iwayama@icot.or.jp

January 18, 1991

1 Introduction

Integrity constraint in logic programming originally relates to checking violation of update [Sadri88]. Consider the following example.

$q \leftarrow m, \neg n$

$r \leftarrow q, p$

p

s

$\leftarrow s, r$

If we add m, then it violates the integrity constraint $\leftarrow s, r$ because both s and r are derived. So, adding of m is prohibited.

Furthermore, recent researches of semantics of logic programming and abduction have revealed that there are another usages of integrity constraint.

1. **Control of Nondeterminism:**
   Nondeterminism of logic programs has been proposed based on multiple stable models by [Sacca90]. Consider the following example.

$pacifist \leftarrow quaker, \neg ab_{pacifist}$

$hawk \leftarrow republican, \neg ab_{hawk}$

$quaker$

$republican$

$ab_{hawk} \leftarrow pacifist$

$ab_{pacifist} \leftarrow hawk$

The above program has the following two stable models:

$$\{quaker, republican, pacifist, ab_{hawk}\}$$

and

$$\{quaker, republican, hawk, ab_{pacifist}\}.$$

This result expresses nondeterminism of logic programs because we can not say which model is better. However, sometimes, we would like to provide preference over those multiple models. In this case, we use integrity constraint which excludes directly unwanted conclusion. By adding

$$\leftarrow hawk$$

we can exclude the latter model and this integrity constraint represents priority of the former model to the latter model.

2. **Computing Abduction**

   In [Satoh90b], we show that a general logic program with integrity constraint can be used to compute abduction. We translate an abductive framework [Kakas90b] to the general logic program with integrity constraint and show one to one correspondence between generalized stable models [Kakas90b] in abductive framework and stable models for the translated program. In abduction, the role of integrity constraint is to exclude some of the subsets of abducibles as non-allowed basic beliefs as [Eshghi89, Kakas90a] pointed out. In the translated program, its role is to exclude some of the multiple stable models of translated program which is not permissible.

So, we believe that a logic program with integrity constraint deserves to be studied.

In [Satoh90b], we provide a bottom-up procedure which calculates stable model for general logic program with integrity constraint. This procedure is based on a procedure

calculating grounded extension of TMS [Satoh90a] and can be regarded as an extension of nondeterministic well-founded bottom-up procedure for calculating stable models for logic programs *without* integrity constraint [Sacca90, Fages90]. In our procedure, we can use integrity constraint of the form $\leftarrow q$ to derive information that $\neg q$ is true and derive p for the rule of the form $p \leftarrow \neg q$. This usage of integrity constraint is *active*, but we use only this type of integrity constraint to derive some facts dynamically but do not use integrity constraint of the form $\leftarrow \neg p$ actively.

The reason why active usage of the latter kind of integrity constraint is important is again related to abduction. We show in [Satoh90b] that if we add $\leftarrow \neg p$ as integrity constraint for the formula p which should be explained then we can show one to one correspondence with explanation in abductive framework and abducibles in stable models for the translated program with the above constraint. In the procedure proposed in [Satoh90b], we have to produce stable models for the program and then check this constraint. So, we might produce unrelated stable models to the formula which should be explained. For example, consider the program which contains $\leftarrow \neg p$ but does not contain a definition of p. Then, the constraint $\leftarrow \neg p$ is never satisfied and therefore, we know that there is no stable model for the program immediately. However,the procedure in [Satoh90b] might produce stable model which do not include p.

In this paper, we provide a new procedure which performs top-down expectation for an integrity constraint of the form $\leftarrow \neg p$ in order to solve the above problem. The idea of this extension is to search a rule which has the possibility of deriving p and if such a rule does not exist, the new procedure immediately fails and if such a rule exists, we select the rule in nondeterministic part of the procedure until we can not select any rules.

The structure of the paper is as follows. In section 2, we give definitions on logic program with integrity constraint. In section 3, we show a new procedure to compute stable models for a logic program with integrity constraint and give some examples. In section 4, we compare this procedure with Eshghi's procedure [Eshghi89].

2 General Logic Program with Integrity Constraint

We follow the definition in [Sadri88] but restrict ourselves to considering propositional case. If we consider predicate case, we change it into ground logic programs by instantiating

every variable to an element of Herbrand universe of considered logic program to obtain propositional program.

Firstly, we define general logic program and integrity constraint.

Definition 1 *Let A be a proposition symbol, and $L_1, ..., L_m (m \geq 0)$ be propositional literals. A* general logic program *consists of (possibly countably infinite) rules of the form:*

$$A \leftarrow L_1, L_2, ..., L_m.$$

We call A the *head* of the rule and $L_1, ..., L_m$ the *body* of the rule. Let R be a rule. We denote the head of R as $head(R)$, the set of positive literals in the body of R as $pos(R)$ and the set of atoms which are obtained by removing negation symbol from negative literals in the body of R as $neg(R)$.

Definition 2 *Let $L_1, ..., L_m (m \geq 0)$ be propositional literals. A set of* integrity constraints *consists of (possibly countably infinite) integrity constraints of the form:*

$$\leftarrow L_1, L_2, ..., L_m.$$

Let C be an integrity constraint. We denote a set of positive literals in C as $pos(C)$ and a set of atoms which are obtained by removing negation symbol from negated atoms in C as $neg(C)$.

We extend the definition of stable models in [Gelfond88] for a general logic program with integrity constraint as follows.

Definition 3 *A general logic program with integrity constraint be a pair $\langle T, I \rangle$ where T is a general logic program and I is a set of integrity constraints. A* stable model *for a general logic program with integrity constraint is a set of propositions M.*

1. *M is equal to the minimal model of T^M where T^M is obtained by the following operation from T. We say that M is a stable model of T.*
   (a) *Deleting every rule R from T that some $N \in neg(R)$ is in M*
   (b) *Deleting every negated atom in the remained rules.*
2. *For every $C \in I$, there is some $P \in pos(C)$ which is not in M or some $N \in neg(C)$ which is in M. We say that M* satisfies *or* does not violate *C and write as $M \models C$.*

This definition gives a stable model of T which satisfies all integrity constraints in I.

3 Computing Stable Models for Logic Program with Integrity Constraint

In this section we give a procedure to compute stable models for general logic program with integrity constraint. At first we give a skeleton of the procedure to show how the procedure works. We will show the detail of the procedure later.

Let (T, I) be a general logic program with integrity constraint.

A Procedure to Compute a Stable Model (skeleton)

$i := 0$,

$M_0, \widetilde{M}_0 := propagate(\emptyset, \emptyset)$.

If $M_0 \cap \widetilde{M}_0 \neq \emptyset$ then **fail**

$R := select\_rule(M_0, \widetilde{M}_0)$

while R is not nil_rule

{i:=i+1,

$M_i, \widetilde{M}_i := propagate(M_{i-1} \cup head(R), \widetilde{M}_{i-1} \cup neg(R))$

If $M_i \cap \widetilde{M}_i \neq \emptyset$ then **fail**

$R := select\_rule(M_i, \widetilde{M}_i)$ }

If there is an integrity constraint C in I s.t. $M_i \not\models C$

then **fail** else **return** M_i.

M_i expresses a set of propositions which is decided to be in the belief set after selecting i rules by *select_rule* and $\widetilde{M}_i$ expresses a set of propositions which is decided to be out of the belief set. And if there is a conflict between M_i and $\widetilde{M}_i$ then M_i is not possible candidate for a stable model. The procedure has non-deterministic choice points in the subprocedure *select_rule*, therefore **fail** in the procedure expresses going back to the recent choice point.

This procedure has two computational directions, one is bottom-up in the subprocedure *propagate* which is already proposed in [Satoh90b], and the other is top-down in the subprocedures *select_rule* and *topdown_check* which are proposed newly in this paper to perform top-down expectation of an integrity constraint.

3.1 Bottom-up Part of the Procedure

Now we review the subprocedure *propagate* from [Satoh90b].

procedure *propagate*$(M_i, \widetilde{M}_i)$

begin

$k := 0$, $M_i^0 := M_i$, $\widetilde{M}_i^0 := \widetilde{M}_i$.

do

$k := k + 1$, $M_i^k := M_i^{k-1}$, $\widetilde{M}_i^k := \widetilde{M}_i^{k-1}$.

For every rule $R = A \leftarrow L_1, L_2, ..., L_m$ in T

1. If $A \notin M_i^{k-1}$ and for every $P \in pos(R), P \in M_i^{k-1}$ and for every $N \in neg(R), N \in \widetilde{M}_i^{k-1}$, then add A to M_i^k.

2. If $A \in \widetilde{M}_i^{k-1}$ and there exists $P \in pos(R)$ s.t. for every $P' \in pos(R)$ except P, $P' \in M_i^{k-1}$, for every $N \in neg(R), N \in \widetilde{M}_i^{k-1}$, then add P to $\widetilde{M}_i^k$.

3. If $A \in \widetilde{M}_i^{k-1}$ and for every $P \in pos(R), P \in M_i^{k-1}$ and for every $N \in neg(R), N \in \widetilde{M}_i^{k-1}$, then **fail.**

For every integrity constraint $C = \leftarrow L_1, L_2, ..., L_m$ in I,

4. If there exists $P \in pos(C)$ s.t. for every $P' \in pos(C)$ except P, $P' \in M_i^{k-1}$, and for every $N \in neg(C), N \in \widetilde{M}_i^{k-1}$, then add P to $\widetilde{M}_i^k$.

5. If for every $P \in pos(C), P \in M_i^{k-1}$ and for every $N \in neg(C), N \in \widetilde{M}_i^{k-1}$, then **fail.**

until $M_i^k = M_i^{k-1}$ **and** $\widetilde{M}_i^k = \widetilde{M}_i^{k-1}$.

return $M_i^k, \widetilde{M}_i^k$

end

propagate performs following jobs.

1. bottom-up construction of the model (by case 1)

2. dynamic checking of the integrity constraint (by cases 3 and 5)

3. active use of the integrity constraint which derive that $\neg q$ is true from the integrity constraint $\leftarrow q$. (by cases 2 and 4)

3.2 Top-down Part of the Procedure

Now we consider how to select a rule in order to start bottom-up construction of the model. In some cases there are reasons why we had better select a specific rule if we want to exclude unrelated models to the formula which should be satisfied. For the integrity constraint, $\leftarrow \neg p$, for example, we know p must be in models immediately. (suppose p is not in any stable model, this means that there is no stable model satisfying the integrity constraint.) So first of all, we had better select a rule which derives p. If such a rule does not exist, we had better select a rule which has a possibility of deriving p.

To explain the possibility of deriving p, we consider the following example.

$$p \leftarrow q, r, \neg s \tag{1}$$

$$q \leftarrow \neg t \tag{2}$$

$$r \leftarrow \neg u \tag{3}$$

Given the integrity constraint, $\leftarrow \neg p$, p must be in models of the above example. Because it is only rule (1) which has p as its head, rule (1) must be used to derive p. In order for p to be in models, q and r must be in by rule (1). We can derive q from rule (2) if we can assume that t is not in models, so rule (2) has a possibility of deriving p. In a similar way, we find rule (3) also has a possibility of deriving p. In this way, we can find a rule with a possibility of deriving p in top-down manner from integrity constraint of the form $\leftarrow \neg p$.

To check whether there is a selectable rule and to decide which rule should be selected, the procedure calls the subprocedure *select_rule*. *select_rule* calls the subprocedure *topdown_check*, which performs top-down expectation. We show these subprocedures as follows.

procedure *select_rule*

If there is a set of proposition Neg satisfying one of the following conditions

1. there exists a rule R in T satisfying the following conditions

   (a) For every $n \in Neg$, $n \in neg(R)$ and $n \notin \widetilde{M}_i$,

   (b) $head(R) \subset \widetilde{M}_i$,

   (c) For every $p \in pos(R), p \in M_i$,

   (d) For every $n \in neg(R), n \notin M_i$.

2. there exists an integrity constraint C in I satisfying the following conditions

   (a) For every $n \in Neg$, $n \in neg(C)$ and $n \notin \widetilde{M}_i$,

   (b) For every $p \in pos(C), p \in M_i$,

   (c) For every $n \in neg(C), n \notin M_i$.

then

$Neg :=$ a set satisfying one of the above conditions
return $topdown\_check(M_i, \widetilde{M}_i, Neg)$

else

select a rule R in T satisfying the following conditions and **return** R.

1. $head(R) \notin M_i$,
2. For every $p \in pos(R), p \in M_i$,
3. For every $n \in neg(R), n \notin M_i$.

If such a rule is not found then **return** nil_rule.

procedure $topdown\_check(M, \widetilde{M}, Neg)$
$M_t, \widetilde{M}_t, M_s := \emptyset$,
$Pos := Neg$
do

select a rule R in T satisfying the following conditions
and if such a rule is not found then **fail**

1. $head(R) \notin M$,
2. $head(R) \notin pos(R)$, $head(R) \notin neg(R)$,
3. $head(R) \in Pos$,
4. For every $p \in pos(R), p \notin \widetilde{M}$, $p \notin \widetilde{M}_t$ and $p \notin M_s$,
5. For every $n \in neg(R), n \notin M$ and $n \notin M_t$.

add $pos(R)$ to M_t

add $neg(R)$ to $\widetilde{M}_t$

add $head(R)$ to M_s

$Pos := pos(R)$

until $Pos \subseteq M$ **return** R.

At first *select_rule* checks whether there is an integrity constraint for which top-down expectation is performed.(precisely, top-down expectation is also performed for some rules.) In *topdown_check*, by M_t and $\widetilde{M}_t$, a rule is selected which is consistent in the rules previously selected during the top-down expectation. Moreover M_s is used to exclude cyclic derivations.

We show that the procedure returns every stable model by an appropriate selection of rules, and it is complete for finite propositional case.

Theorem 1 *Let $\langle T, I \rangle$ be a logic program with integrity constraint.*

1. *If the procedure outputs M, then M is a stable models for $\langle T, I \rangle$.*
2. *If T and I are finite, then the procedure outputs all stable models by exhaustive search.*

Proof: See Appendix.

3.3 Examples

We compare our procedure with the procedures of [Sacca90, Satoh90b]. The following example shows the difference.

Example 1 *Difference of Three Procedures*

Consider the example in Introduction again:

$$pacifist \leftarrow quaker, \neg ab_{pacifist} \quad (1)$$

$$hawk \leftarrow republican, \neg ab_{hawk} \quad (2)$$

$$quaker \quad (3)$$

$$republican \quad (4)$$

$$ab_{hawk} \leftarrow pacifist \quad (5)$$

$$ab_{pacifist} \leftarrow hawk \tag{6}$$

and the integrity constraint:

$$\leftarrow hawk \tag{7}$$

The procedure of [Sacca90] produces next two stable models stated in Introduction for a logic program (1)~(6).

$$\{quaker, republican, pacifist, ab_{hawk}\},$$

$$\{quaker, republican, hawk, ab_{pacifist}\}.$$

So, this process has nondeterminism of producing two stable models. Then, we discard the latter because that model does not satisfy the integrity constraint (7).

Next we show the execution of the procedure of [Satoh90b] with contents of M_i and $\widetilde{M}_i$.

0. $M_0 = \{quaker, republican\}, \widetilde{M}_0 = \{hawk\}$,
   because from (3) and (4), *quaker* and *republican* must be in M_0 by case 1 in *propagate* respectively, and from (7), *hawk* must be in $\widetilde{M}_0$ by case 4 in *propagate*.

1. Select rule (1). Then, $M_1 = \{quaker, republican, pacifist, ab_{hawk}\}, \widetilde{M}_1 = \{hawk, ab_{pacifist}\}$.

2. Since there is no selected rule, M_1 is returned.

Though our procedure results in the selection of rule (1) as the procedure of [Satoh90b], our procedure decides to select rule (1) by top-down expectation. The execution of our procedure is as follows.

0. $M_0 = \{quaker, republican\}, \widetilde{M}_0 = \{hawk\}$,
   by *propagate*.

1. $Neg = \{ab_{hawk}\}$ in *select_rule*. Select rule (5) and (1) in *topdown_check*. Then, rule (1) is returned by *topdown_check* because $quaker \in M_0$. So, by *propagate*, $M_1 = \{quaker, republican, pacifist, ab_{hawk}\}, \widetilde{M}_1 = \{hawk, ab_{pacifist}\}$.

2. Since there is no selected rule, M_1 is returned.

In [Satoh90b], we have shown that a general logic program with integrity constraint can be used to compute abduction. We translate an abductive framework [Kakas90b] to the

general logic program with integrity constraint. By next example we show the process to compute abduction with our procedure.

Example 2 *Abduction [Satoh90b]*

Consider the following logic program T [Kakas90b, p.387]:

$$p \leftarrow b \tag{1}$$

$$q \leftarrow a \tag{2}$$

with abducibles $A = \{a, b\}$,

and the following set of integrity constraints I:

$$\leftarrow q, b \tag{3}$$

$$\leftarrow \neg q, \neg b \tag{4}$$

and suppose an observation q is given.

Translation from this abductive framework by [Satoh90b] is follows. We add the following rules, $\Gamma(A)$, to the above logic program.

$$a \leftarrow \neg \tilde{a} \tag{5}$$

$$\tilde{a} \leftarrow \neg a \tag{6}$$

$$b \leftarrow \neg \tilde{b} \tag{7}$$

$$\tilde{b} \leftarrow \neg b \tag{8}$$

And we add the following integrity constraint to the above integrity constraint.

$$\leftarrow \neg q \tag{9}$$

To compute abductive explanation for the abductive framework $\langle T, A, I \rangle$, we compute stable models for the translated logic program $\langle T \cup \Gamma(A), I \cup \{\leftarrow \neg q\} \rangle$.

The execution of our procedure for $\langle T \cup \Gamma(A), I \cup \{\leftarrow \neg q\} \rangle$ is as follows.

0. $M_0 = \emptyset, \widetilde{M}_0 = \emptyset$,

1. $Neg = \{q\}$ in *select_rule* because of rule (9). Select rule (2) and (5) in *topdown_check*. Then, rule (5) is returned by *topdown_check* because rule (5) has no positive proposition. So, by *propagate*, $M_1 = \{a, q, \tilde{b}\}, \widetilde{M}_1 = \{\tilde{a}, b\}$.

2. Since there is no selected rule, M_1 is returned.

So we get a abductive explanation $M_1 \cap A = \{a\}$.

In the above execution of our procedure we can calculate a stable model without backtracking thanks to top-down expectation of integrity constraint. If we use the procedure of [Satoh90b] to compute stable models for the translated logic program, then we have six alternatives to select a rule.

4 Related Work

[Eshghi89] shows the abduction procedure to calculate explanation for a give observation, which is a generalization of SLDNF. [Kakas90a] extends the top-down procedure in [Eshghi89] in order to manipulate arbitrary hypothesis. Though these procedures originally compute abduction, these procedures can compute models for general logic program if that general logic program contains no abducible. Now we compare our procedure with their procedure.

In their framework, negative literals in the program are translated into abducibles, which are positive literals. Their procedures perform SLD when an ordinary (positive) literal is selected, when an abducible (expressing the negative literal) is selected they do a generalized Negation by Failure. So they always compute in a top-down manner whichever the selected literal is positive or negative. On the other hand, our procedure computes mainly in a bottom-up manner except that it computes in a top-down manner for integrity constraint of the form $\leftarrow \neg p$. We incorporate the top-down nature of SLD in [Eshghi89]'s procedure into our procedure for the efficiency. While the procedure in [Eshghi89] is not sound for any general logic programs, our procedure is sound for any programs. After a refutation is produced in [Eshghi89]'s procedure if the procedure confirms in bottom-up manner that rules which are not used in the refutation are consistent in the derived explanation, then [Eshghi89]'s procedure may become sound for any programs.

Appendix

Proof of Theorem 1:

We can show theorem 1 by the adaptation of the proof that the procedure of [Satoh90b] is correct.

Consider the following simple procedure to compute a stable model.

Let $\langle T, I \rangle$ be a general logic program with integrity constraint.

A Simple Procedure to Compute a Stable Model

$i := 0$

Step 1:

Select a rule $R_i = A_i \leftarrow L_1, L_2, ..., L_m$ in T satisfying the following conditions and go to **Step 2**.

1. $A_i \notin M_i$,
2. For every $P \in pos(R_i), P \in M_i$,
3. For every $N \in neg(R_i), N \notin M_i$.

If such a rule is not found and there is an integrity constraint C in I s.t. $M_i \not\models C$ then **fail** else **return** M_i.

Step 2:

$i := i + 1$,
$M_i = M_{i-1} \cup \{A_{i-1}\}$
If there exists $R_k (0 \leq k \leq i-1)$ such that for some $N \in neg(R_k), N \in M_i$
then **fail** else go to **Step 1**.

We denote our procedure in section 3 as $proc(O)$ and denote the above simple procedure as $proc(S)$. We need the following two lemmas.

Lemma 1

1. *If there is a selection of rules such that proc(O) outputs M, then there is a selection of rules such that proc(S) also outputs M.*
2. *If T and I are finite and there is a selection of rules such that proc(S) outputs M, then there is a selection of rules such that proc(O) also outputs M.*

Lemma 2

1. *If proc(S) outputs M, then M is a stable models for $\langle T, I \rangle$.*

2. If T and I are finite, then proc(S) outputs all stable models by an exhaustive search.

We can show Lemma 2 by extending the correspondence between stable model and grounded model from [Elkan90, Theorem3.8].

Proof of Lemma 1:

1. Suppose $proc(O)$ outputs M with a selection of rules $R_0, ..., R_n$ where $R_i (0 \leq i \leq n)$ is used at the while sentence in the main procedure or case 1 in the subprocedure *propagate*. We can show this sequence of rule can be used in $proc(S)$ to output M.

2. Let $\langle T, I \rangle$ be a finite general logic program with finite integrity constraint. Suppose $proc(S)$ outputs M with a selection of rules $R_0, ..., R_n$. We show that we can modify this selection of rules such that:

   1. $proc(S)$ still outputs M with a modified selection of rules,

   2. $proc(O)$ also outputs M with the modified selection of rules.

   To show the above, we show the following condition is true for every i in the main procedure of $proc(O)$ and every k in the subprocedure *propagate* of $proc(O)$.

   1. We can modify the above selection of rules up to given i and k, such that:

      (a) $proc(S)$ still outputs M with a modified selection of rules,

      (b) $proc(O)$ can use the modified selection of rules up to i and k,

      (c) M_i^k is a subset of M.

   2. M and $\widetilde{M}_i^k$ are mutually exclusive.

We can prove this by induction of i and k in $proc(O)$. Then since every stable model must be finite, $proc(O)$ eventually terminates with output M. □

Theorem 1 is proved by Lemma 1 and Lemma 2. □

References

[Elkan90] Elkan, C., A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems, *Artificial Intelligence*, **43**, pp. 219 – 234 (1990).

[Eshghi89] Eshghi, K., Kowalski, R. A., Abduction Compared with Negation by Failure, *Proc. of ICLP'89*, pp. 234 – 254 (1989).

[Fages90] Fages, F., A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics, *Proc. of ICLP'90*, pp. 442 – 458 (1990).

[Gelfond88] Gelfond, M., Lifschitz, V., The Stable Model Semantics for Logic Programming, *Proc. of LP'88*, pp. 1070 – 1080 (1988).

[Kakas90a] Kakas, A. C., Mancarella, P., On the relation between Truth Maintenance and Abduction, *Proc. of 3rd Nonmonotonic Reasoning Workshop*, pp. 158 – 176 (1990).

[Kakas90b] Kakas, A. C., Mancarella, P., Generalized Stable Models: A Semantics for Abduction, *Proc. of ECAI'90*, pp. 385 – 391 (1990).

[Sacca90] Sacca, D., Zaniolo, C., Stable Models and Non-Determinism in Logic Programs with Negation, *Proc. of PODS'90*, pp. 205 – 217 (1990).

[Sadri88] Sadri, F., Kowalski, R. A., A Theorem-Proving Approach to Database Integrity, *Foundations of Deductive Databases and Logic Programming*(J. Minker, ed.), Morgan Kaufmann, Los Altos, Ca., pp. 313 – 362 (1988).

[Satoh90a] Satoh, K., Iwayama, N., Sugino, E., Konolige, K., An Implementation of TMS in Concurrent Logic Programming Language: Preliminary Report, *ICOT–TR–568*, ICOT (1990).

[Satoh90b] Satoh, K., Iwayama, N., Computing Abduction by Using the TMS, submitted for publication (1990).