ICOT Technical Report: TR-739

TR-739

A Correct Top-Down Proof Procedure for a General Logic Program with Integrity Constraints

by
K. Satoh & N. Iwayama

February, 1992

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 Correct Top-Down Proof Procedure for a General Logic Program with Integrity Constraints

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

January 28, 1992

Abstract

We present a correct top down procedure for every general logic program with integrity constraints. Recently, stable model semantics [4] of logic programs was proposed and has been investigated intensively. Although there are correct bottom-up procedures for every general logic program [10, 3, 12] based on stable models, there are no proposed correct top-down procedure for every general logic program. Our proposed procedure is correct not only for successful derivation but also for finite failure. This procedure is an extension of Eshghi's procedure [2] which is correct for every call-consistent logic program, and can be regarded as a combination of the ancestor-resolution [8] and consistency checking in updates of implicit deletions [11].

1 Introduction

Recently, stable model semantics was proposed for an alternate semantics of general logic programs [4]. The semantics reflects epistemic character of negation in logic programming and is related to the existing systems of nonmonotonic reasoning such as Default Logic and Autoepistemic Logic. Moreover, the semantics can be used as a formal basis of abduction in logic programming [2, 5]. So, the semantics is very useful for representing various non-deductive reasoning in terms of logic programming.

However, there is a problem of computation for stable model semantics. Although some have proposed bottom-up procedures which compute stable models for every logic program [10, 3, 12], only Eshghi and Kowalski [2] proposed a top-down procedure which is not correct for every logic program but a call-consistent logic programs. Major difficulty of top-down procedure comes from its locality whereas stable model semantics has a global property. In general, we must search every rule in a logic program to obtain a stable model. Top-down procedure, however, searches only rules relevant for a query.

To solve the problem, some people are working on a modification of semantics which fits to Eshghi and Kowalski's procedure [7, 6]. Although this approach seems interesting, there are still arguments which semantics is best. In stead of this approach, this paper goes into the other direction. That is, we stick to the original stable model semantics and try to fix Eshghi's procedure.

A solution from this approach is as follows. Firstly, we guarantee that a considered logic program has always a stable model by checking consistency in accordance with addition of rules. Secondly, we pursue a procedure which answers "yes" (if stopped) if there is a stable model which satisfies a query and answers "no" (if stopped) if there is no stable model which satisfies a query. In our opinion, Eshghi's procedure has the problem on correctness because it does not search rules which are actually relevant to query.

Consider the following example in [2].

$$p \leftarrow \sim q \tag{1}$$
$$q \leftarrow \sim p \tag{2}$$
$$r \leftarrow q \tag{3}$$
$$r \leftarrow \sim r \tag{4}$$

If we ask $\leftarrow p$ then Eshghi's procedure answers "yes" since the procedure only considers rules (1) and (2). However, if p is true, then q must be false and the rule (3) is no more than applicable and it leads to inconsistency by the rule (4). As far as we know, this phenomenon was firstly observed by Sadri [11] and called "implicit deletion". Sadri also gave a procedure which checks integrity constraints by implicit deletion and proved its correctness. We use Sadri's idea to infer that rules (3) and (4) are also relevant to the query $\leftarrow p$.

Then, our procedure becomes a combination of ancestor resolution [8] and consistency check adapted from [11]. Concerning ancestor resolution, we use it in a different manner and for a different purpose. We use ancestor resolution in a different manner because rules (clauses) in a logic program is directed and we do not consider its contraposi-

tives. We use ancestor resolution for a different purpose because we want to detect consistency instead of inconsistency.

Moreover, by obtaining this procedure, we automatically obtain a procedure for our first purpose, to check consistency for added rules. It is because this procedure contains a subprocedure which checks consistency for rules relevant to query.

The structure of the paper is as follows. Firstly, we give definitions necessary for our procedure and then we show a top-down procedure of evaluating a query for a general logic program with integrity constraints. Finally, we compare it with related researches.

2 Top-down procedure

We restrict ourselves to considering a propositional case. If we consider predicate case, we change it into a ground logic program by instantiating every variable to an element of Herbrand universe of considered logic program to obtain a propositional program.

A literal is a proposition or negated proposition of the form $\sim l$.

Definition 1 *Let l be a literal. We denote the* inverse *of the literal as $\overline{l}$ and define it as follows.*

1. *If l is a positive literal, then $\overline{l} = \sim l$.*
2. *If l is a negative literal of the form $\sim l'$, then $\overline{l} = l'$.*

We use special literals $\bot$ and $\top$ where $\bot$ expresses inconsistency and $\top = \overline{\bot}$.

Then, we define a general logic program and integrity constraints.

Definition 2 *Let A be a propositional 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 $head(R)$ and the set of literals in the body of R as $body(R)$ and the set of positive literals in the body of R as $pos(R)$ and $neg(R) = \{p | \sim p \in body(R)\}$

Definition 3 *Let $L_1, ..., L_m (m \geq 0)$ be propositional literals. A set of* integrity constraints *consists of (possibly countably infinite) formulas of the form:*

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

$\perp$ means inconsistency and we write integrity constraints as the above form for notational convenience so that we do not have to distinguish integrity constraints and rules. So, from this point, we call the above set of rules and integrity constraints *a logic program.*

Definition 4 *A* stable model *for a logic program is a set of propositions M if $\perp \notin M$ and M is equal to the minimal model of T^M where*

$$T^M = \{R | head(R) = head(R') \text{ and } pos(R) = pos(R') \text{ and } neg(R) = \emptyset \text{ where } R' \in T \text{ and } neg(R') \cap M = \emptyset\}.$$

This definition gives a stable model of T which satisfies all integrity constraints. We say that T is *consistent* if there exists a stable model for T.

Before showing a procedure, we need the following definitions.

Definition 5 *Let T be a logic program.*

1. *A set of resolvents w.r.t. a positive literal p and T, $resolve(p, T)$ is a set of such a rule R that:*
   $head(R) = head(R')$ *and* $body(R) = body(R') - \{p\}$
   where $R' \in T$ *and* $p \in body(R')$.

2. *A set of resolvents w.r.t. a negative literal $\sim p$ and T, $resolve(\sim p, T)$ is a set of such a rule R that:*

   (a) *either* $head(R) = \perp$ *and* $body(R) = body(R')$
   where $R' \in T$ *and* $head(R') = p$,

   (b) *or* $head(R) = head(R')$ *and* $body(R) = body(R') - \{\sim p\}$
   where $R' \in T$ *and* $\sim p \in body(R')$.

The second definition of a resolvent for a negative literal corresponds with a resolvent obtained by "extended" resolution introduced in [11]. This extended resolution and the resolution for a positive literal expresses "forward" evaluation of the rule.

Definition 6 *Let l be a literal and T be a logic program. Then, a set of deleted rules, $del(l, T)$, w.r.t. l and T is such a rule R that $R \in T$ and $\overline{l} \in body(R)$.*

Now, we give a derivation procedure. It consists of 4 subprocedures, $derive(p, \Delta)$, $literal\_con(l, \Delta)$, $rule\_con(R, \Delta)$ and $deleted\_con(p, \Delta)$ where p is a proposition and Δ is a set of literals which have already been derived and l is a literal and R is a rule.

Each subprocedure returns a set of literals which is a union of Δ and literals which are newly found to be derived during the execution of subprocedure.

In subprocedures, there is a **select** operation and **fail** operation. A **select** operation expresses a nondeterministic choice among alternatives and a **fail** operation expresses an immediately failure of the execution. So, a subprocedure succeeds if calls of subprocedures in the subprocedure is successful. We say *a subprocedure succeeds with* Δ if the subprocedure successfully returns Δ.

An informal specification of 4 procedures as follows.

1. $derive(p, \Delta)$ mainly searches a rule of p whose body is true. Intuitively speaking, if this procedure succeeds, there exists a stable model which satisfies such a rule.

2. $literal\_con(l, \Delta)$ checks consistency of l. Intuitively speaking, if this procedure succeeds, there exists a stable model which satisfies l. In this procedure, we assume l for ancestor resolution.

3. $rule\_con(R, \Delta)$ checks consistency of a rule R. Intuitively speaking, if this procedure succeeds, there exists a stable model which satisfies R. This procedure can be used also for checking integrity constraints for addition of a rule.

4. $deleted\_con(p, \Delta)$ checks consistency of implicit deletions of rules related to p.

$derive(p, \Delta)$ p: proposition; Δ: set of literals
begin
  if $p \in \Delta$ **then return** Δ
  elseif $\bar{p} \in \Delta$ **then fail**
  else
  begin
    select $R \in T$ such that $head(R) = p$
    if such a rule is not found **then fail**
    $\Delta_0 := \Delta$, $i := 0$
    for every $l \in body(R)$ **do**
    begin
      if l is positive and $derive(l, \Delta_i)$ succeeds with Δ_{i+1}
        then $i := i + 1$ and **continue**
      elseif l is negative and $literal\_con(l, \Delta_i)$ succeeds with Δ_{i+1}
        then $i := i + 1$ and **continue**
    end
    if $literal\_con(p, \Delta_i)$ succeeds with Δ' **then return** Δ'
  end
end ($derive$)

```
literal_con(l, Δ) l: literal; Δ: set of literals
begin
    if l ∈ Δ then return Δ
    elseif l̄ ∈ Δ then fail
    else
    begin
      Δ_0 := {l} ∪ Δ, i := 0
      for every R ∈ resolve(l, T) do
        if rule_con(R, Δ_i) succeeds with Δ_{i+1}
          then i := i + 1 and continue
      for every p ∈ {head(R)|R ∈ del(l, T)} do
        if deleted_con(p, Δ_i) succeeds with Δ_{i+1}
          then i := i + 1 and continue
    end
    return Δ_i
end (literal_con)

rule_con(R, Δ) R: rule; Δ: set of literals
begin
    select (a) or (b)
    (a) select l ∈ body(R)
      if l is positive and literal_con(l̄, Δ) succeeds with Δ'
        then return Δ'
      elseif l is negative and derive(l̄, Δ) succeeds with Δ'
        then return Δ'
    (b) Δ_0 = Δ, i := 0
      for every l ∈ body(R) do
      begin
        if l is positive and derive(l, Δ_i) succeeds with Δ_{i+1}
          then i := i + 1 and continue
        elseif l is negative and literal_con(l, Δ_i) succeeds with Δ_{i+1}
          then i := i + 1 and continue
      end
      if literal_con(head(R), Δ_i) succeeds with Δ' then return Δ'
end (rule_con)

deleted_con(p, Δ) p: proposition; Δ: set of literals
begin
    select (a) or (b)
    (a) if derive(p, Δ) succeeds with Δ' then return Δ'
    (b) if literal_con(~p, Δ) succeeds with Δ' then return Δ'
end (deleted_con)
```

If we remove *deleted_con* and do not consider resolvents obtained by the "forward" evaluation of the rule, then this procedure coincides with Eshghi's procedure [2]. In other words, our procedure is obtained by augmenting Eshghi's procedure by an integrity constraint checking in a bottom-up manner and an implicit deletion checking.

We can show the following theorems.

Theorem 1 *Let T be a consistent logic program. If $derive(p, \{\top\})$ succeeds, then there exists a stable model M for T such that $M \models p$.*

This theorem means that if the procedure $derive(p, \{\top\})$ answers "yes", then there is a stable model which satisfies a query $\leftarrow p$.

The following theorem expresses an integrity constraint check for an added rule.

Theorem 2 *Let T be a consistent logic program and R be a rule. If $rule\_con(R, \{\top\})$ succeeds, then $T \cup \{R\}$ is consistent.*

So, by this theorem, we can use the above procedure to check consistency in accordance with addition of rules.

The following are theorems related to the correctness for finite failure.

Theorem 3 *Let T be a logic program. Suppose that every selection of rules terminates for $derive(p, \{\top\})$. If there exists a model M for T such that $M \models p$, then there is a selection of rules such that $derive(p, \{\top\})$ succeeds.*

This theorem means that if we can do an exhaustive search for selection of rules and there is a stable model which satisfies a query, then the procedure always answers "yes".

From the above theorem, we obtain the following corollary for a finite failure.

Corollary 1 *Let T be a logic program. If $derive(p, \{\top\})$ fails, then for every stable model M for T, $M \not\models p$.*

The procedure $derive(p, \{\top\})$ answers "no", then there is no stable model which satisfies the query. Moreover, the above corollary means that we can use a finite failure to check if a literal is true in all stable models since finite failure of $derive(p, \{\top\})$ means that every stable model satisfies $\sim p$. From the above correctness for a finite failure, we can say completeness result for a finite program provided that we can check a loop of the same call of $derive(p, \Delta)$ to make the execution fail.

Corollary 2 *Let T be a finite logic program. If there exists a stable*

model M for T such that $M \models p$, derive$(p, \{\top\})$ (with loop check) succeeds.

Corollary 3 *Let T be a finite consistent logic program. If there exists a stable model M for T such that $M \models R$, there is a selection of rules such that rule_con$(R, \{\top\})$ (with loop check) succeeds.*

3 Examples

Example 1 *Consider the following program.*

$$p \leftarrow \sim q \quad (1)$$
$$q \leftarrow \sim p \quad (2)$$
$$r \leftarrow q \quad (3)$$
$$r \leftarrow \sim r \quad (4)$$

Then, Figure 1 shows a sequence of calling procedures obtained for derive$(q, \{\top\})$.

Example 2 *Figure 2 shows a sequence of calling procedures obtained for derive$(p, \{\top\})$ for the above program by depth-first search. (We omit the success message so that the sequence is shown in one page.)*

In Figure 2, since *deleted_con*$(r, \{p, \sim q, \top\})$ fails, we can detect inconsistency with respect to r and come to know that assuming p and $\sim q$ is impossible.

4 Related Work

4.1 Eshghi's procedure

As we have said in the section of the top-down procedure, if we remove *deleted_con* and *literal_con* for positive literals from the procedure and do not consider resolvents obtained by the "forward" evaluation of the rule, it is identical with Eshghi's procedure. Although *literal_con* for positive literals and "forward" evaluation of the rule is mainly used for a forward checking of integrity constraint, note that it is also essential if we consider consistency by implicit deletion since we must check which rule is deleted by assuming new literals in a bottom-up manner. Therefore, the overall procedure is needed to check consistency not only for a general logic program with integrity constraints but also for a general logic program *without* integrity constraints.

$derive(q, \{\top\})$
select (2)
  $literal\_con(\sim p, \{\top\})$
    $rule\_con((\bot \leftarrow \sim q), \{\sim p, \top\})$ (resolvent with (1))
      $derive(q, \{\sim p, \top\})$
        select (2)
          $literal\_con(\sim p, \{\sim p, \top\})$
          succeeds with $\{\sim p, \top\}$ ($literal\ con(\sim p, \{\sim p, \top\})$)
        $literal\_con(q, \{\sim p, \top\})$
          $rule\_con((r \leftarrow), \{q, \sim p, \top\})$ (resolvent with (3))
            $literal\_con(r, \{q, \sim p, \top\})$
              $deleted\_con(r, \{r, q, \sim p, \top\})$ (for (4))
                $derive(r, \{r, q, \sim p, \top\})$
                succeeds with $\{r, q, \sim p, \top\}$ ($derive(r, \{...\})$)
              succeeds with $\{r, q, \sim p, \top\}$ ($deleted\_con(r, \{...\})$)
            succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(r, \{q, \sim p, \top\})$)
          succeeds with $\{r, q, \sim p, \top\}$ ($rule\_con((r \leftarrow), \{q, \sim p, \top\})$)
          $deleted\_con(p, \{r, q, \sim p, \top\})$ (for (1))
            $literal\_con(\sim p, \{r, q, \sim p, \top\})$ (for (1))
            succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(\sim p, \{...\})$)
          succeeds with $\{r, q, \sim p, \top\}$ ($deleted\_con(p, \{r, q, \sim p, \top\})$)
        succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(q, \{\sim p, \top\})$)
      succeeds with $\{r, q, \sim p, \top\}$ ($derive(q, \{\sim p, \top\})$)
    succeeds with $\{r, q, \sim p, \top\}$ ($rule\_con((\bot \leftarrow \sim q), \{\sim p, \top\})$)
    $rule\_con((q \leftarrow), \{r, q, \sim p, \top\})$ (resolvent with (2))
      $literal\_con(q, \{r, q, \sim p, \top\})$
      succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(q, \{r, q, \sim p, \top\})$)
    succeeds with $\{r, q, \sim p, \top\}$ ($rule\_con((q \leftarrow), \{r, q, \sim p, \top\})$)
  succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(\sim p, \{\top\})$)
$literal\_con(q, \{r, q, \sim p, \top\})$
succeeds with $\{r, q, \sim p, \top\}$ ($literal\_con(q, \{r, q, \sim p, \top\})$)
succeeds with $\{r, q, \sim p, \top\}$ ($derive(q, \{\top\})$)

Figure 1: Calling Sequence for $derive(q, \{\top\})$

$derive(p, \{\top\})$
select (1)
  $literal\_con(\sim q, \{\top\})$
    $rule\_con((p \leftarrow), \{\sim q, \top\})$ (resolvent with (1))
      $literal\_con(p, \{\sim q, \top\})$
        $deleted\_con(q, \{p, \sim q, \top\})$ (for (2))
          $literal\_con(\sim q, \{p, \sim q, \top\})$ (for (2))
    $rule\_con((\bot \leftarrow \sim p), \{p, \sim q, \top\})$ (resolvent with (2))
      $derive(p, \{p, \sim q, \top\})$
    $deleted\_con(r, \{p, \sim q, \top\})$ (for (3))
      $derive(r, \{p, \sim q, \top\})$
        select (3)
          $derive(q, \{p, \sim q, \top\})$
          fail $(derive(q, \{p, \sim q, \top\}))$
        select (4)
          $literal\_con(\sim r, \{p, \sim q, \top\})$
            $rule\_con((\bot \leftarrow q), \{\sim r, p, \sim q, \top\})$ resolvent with (3)
              $literal\_con(\sim q, \{\sim r, p, \sim q, \top\})$
            $rule\_con((\bot \leftarrow \sim r), \{\sim r, p, \sim q, \top\})$ resolvent with (4)
              $derive(r, \{\sim r, p, \sim q, \top\})$
              fail $(derive(r, \{\sim r, p, \sim q, \top\}))$
              $literal\_con(\sim r, \{\sim r, p, \sim q, \top\})$
              $literal\_con(\bot, \{\sim r, p, \sim q, \top\})$
              fail $(literal\_con(\bot, \{\sim r, p, \sim q, \top\}))$
            fail $(rule\_con((\bot \leftarrow \sim r), \{\sim r, p, \sim q, \top\}))$
          fail $(literal\_con(\sim r, \{p, \sim q, \top\}))$
      fail $(derive(r, \{p, \sim q, \top, \}))$
      $literal\_con(\sim r, \{p, \sim q, \top\})$
        $rule\_con((\bot \leftarrow q), \{\sim r, p, \sim q, \top\})$ resolvent with (3)
          $literal\_con(\sim q, \{\sim r, p, \sim q, \top\})$
        $rule\_con((\bot \leftarrow \sim r), \{\sim r, p, \sim q, \top\})$ resolvent with (4)
          $derive(r, \{\sim r, p, \sim q, \top\})$
          fail $(derive(r, \{\sim r, p, \sim q, \top\}))$
          $literal\_con(\sim r, \{\sim r, p, \sim q, \top\})$
          $literal\_con(\bot, \{\sim r, p, \sim q, \top\})$
          fail $(literal\_con(\bot, \{\sim r, p, \sim q, \top\}))$
        fail $(rule\_con((\bot \leftarrow \sim r), \{\sim r, p, \sim q, \top\}))$
      fail $(literal\_con(\sim r, \{p, \sim q, \top\}))$
    fail $(deleted\_con(r, \{p, \sim q, \top\}))$
  fail $(literal\_con(\sim q, \{\top\}))$
fail $(derive(p, \{\top\}))$

Figure 2: Calling Sequence for $derive(p, \{\top\})$

4.2 Ancestor Resolution

literal_con actually corresponds with ancestor resolution [8] in the sense that *literal_con* firstly searches to check if there is the checked literal in the assumed literals and then, to check if the checked literal is inconsistent with the checked literal being added to a set of the assumed literals.

However, there are three main differences between ancestor resolution and *literal_con*.

1. We do not want inconsistency which is the original goal for ancestor resolution, but we want consistency to assume the checked literal.

2. In ancestor resolution, a contrapositive form of a rule is logically equivalent to the original form, but in logic programming, it is not the case. Consider the following three rules.

   $r \leftarrow \sim p$ (1)
   $p \leftarrow \sim q$ (2)
   $q \leftarrow \sim p$ (3)

   If we consider a program $T_1 = \{(1),(2)\}$ and a query $\leftarrow r$ for T_1, then we check consistency for $\sim p$ and then fails since p is derived by the rule (2). On the other hand, if we consider a program $T_2 = \{(1),(3)\}$ and a query $\leftarrow r$ for T_2, then we check consistency for $\sim p$ and then succeeds since q must be true from the rule (3) but q is consistent. This example shows that if we use a contrapositive form, the result might be different. So, we can say that we have a *directed* ancestor resolution in our procedure.

3. In ancestor resolution, we discard assumed literals in a branch of an AND-tree if we detect inconsistency. On the other hand, we accumulate assumed literals in Δ in our procedure so that coherence of assumed literals maintains over branches of an AND-tree.

4.3 Sadri's Integrity Constraint Checker

Sadri and Kowalski propose an integrity constraint check by augmenting SLDNF procedure with "forward" evaluation of rules and an inconsistency check for implicit deletions [11]. Although they show correctness for their procedure, they only show completeness for a logic program with integrity constraints which contain no negative literals in

the body of each rule. So, they does not guarantee consistency for an addition of a rule for every general logic program. On the other hand, if $rule\_con(R, \{\top\})$ succeeds, we can guarantee that R is consistent with the current program even if it contains negative literals in the body.

Moreover, since we use ancestor resolution, we can prove consistency for addition of rules for a wider class of logic programs. For example, consider the following program:

$$p \leftarrow \sim q \quad (1)$$
$$q \leftarrow \sim p \quad (2)$$

If we check consistency for addition of p in Sadri's system, we invoke a query $\leftarrow p$ and see if it finitely fails. However, it enters an infinite loop and does not stop, although $rule\_con(p, \{\top\})$ succeeds in showing consistency for the addition.

4.4 Poole's System

Poole [9] develops default and abductive reasoning system called *Theorist*. Since our procedure uses assumptions, his work is also related. However, there are two major differences.

1. The basic language for *Theorist* is a first-order language whereas we use a logic program. So, in *Theorist*, a contrapositive form of a rule is logically equivalent to the rule whereas in our setting, it is not always true.

2. Assumptions in *Theorist* correspond with normal defaults without prerequisites in Default Logic, whereas in our setting, rules in a logic program can be regarded as arbitrary defaults. So, our procedure is a kind of proof procedure for a default theory which consists of only arbitrary default rules without proper axioms.

5 Conclusion

In this paper, we propose a top-down procedure for a general logic program with integrity constraints. This procedure can be regarded as a combination of a modified ancestor resolution and checking of consistency by an implicit deletion. As a further study, we think that the following are needed

1. Negations in general logic programs are "negation as ignorance". However, in nonmonotonic reasoning, we use not only negation

as ignorance but also a logical negation. So, we should handle those logical negations as well in our procedure.

2. We should compare with bottom-up procedures to compute stable models [10, 3, 12] in terms of computational complexity.

Appendix

We need the following definitions and lemmas to prove Theorems.

Definition 7 *Let Δ be a set of literals. $\mathcal{F}(\Delta)$ is a set of rules defined as:*

1. $p \leftarrow$ *if* $p \in \Delta$ *and* $p \neq \top$, *or*
2. $\bot \leftarrow p$ *if* $\sim p \in \Delta$.

Definition 8 *Let T be a logic program. A set of propositions M is a* finite grounded model *for T if the following are satisfied.*

1. *M is a model of T.*
2. *M can be written as a sequence of propositions $\langle p_1, p_2, ..., p_n \rangle$ such that each p_j has at least one rule R_j such that $head(R_j) = p_j$ and $pos(R_j) \subseteq \{p_1, ..., p_{j-1}\}$ where $p_1, ..., p_{j-1}$ are the element of the sequence up to $j-1$ and $(neg(R_j) \cap M) = \emptyset$. We say a sequence of such rules for proposition p_j is* a sequence of supporting rules *for p_j and especially a sequence of supporting rules for p_n, $\langle R_1, R_2, ...R_n \rangle$. is* a sequence of supporting rules *for M.*

We can prove the following lemma by extending [1, Theorem 3.8].

Lemma 1 *Let T be a logic program. A set of propositions M is a finite grounded model for T if and only if M is a finite stable model for T.*

Lemma 2 *Let T be a logic program and Δ be a set of literals such that it includes $\top$ and for every $l \in \Delta$, $\overline{l} \notin \Delta$.*

1. *Suppose $derive(p, \Delta)$ succeeds with Δ' and let $\mathcal{R}$ be a set of rules in T which are checked during the execution. Then, $pos(\Delta')$ (a set of positive literals in Δ') is a stable model for $\mathcal{R} \cup \mathcal{F}(\Delta)$.*
2. *Suppose $literal\_con(l, \Delta)$ succeeds with Δ' and let $\mathcal{R}$ be a set of rules in T which are checked during the execution. Then, $pos(\Delta')$ is a stable model for $\mathcal{R} \cup \mathcal{F}(\{l\} \cup \Delta)$.*

3. *Suppose* $rule\_con(R, \Delta)$ *succeeds with* Δ' *and let* $\mathcal{R}$ *be a union of* $\{R\}$ *and a set of rules in* T *which are checked during the execution. Then,* $pos(\Delta')$ *is a stable model for* $\mathcal{R} \cup \mathcal{F}(\Delta)$.

4. *Suppose* $delete\_con(p, \Delta)$ *succeeds with* Δ' *and let* $\mathcal{R}$ *be a set of rules in* T *which are checked during the execution. Then,* $pos(\Delta')$ *is a stable model for* $\mathcal{R} \cup \mathcal{F}(\Delta)$.

Proof of Lemma: By induction of the number of calls of subprocedures during the execution. □

Proof of Theorem 1: Suppose $derive(p, \{\top\})$ succeeds with Δ but for every stable model M for T, $M \not\models \mathcal{F}(\Delta)$. Let $\mathcal{R}$ be a set of checked rules. Let $\mathcal{R}_1$ be a set of rules in $\mathcal{R}$ with a literal l such that $l \notin \Delta$ and $\overline{l} \notin \Delta$ and let $\mathcal{R}_2$ be $\mathcal{R} - \mathcal{R}_1$. Then, every rule in $\mathcal{R}_1$ satisfies the following conditions.

1. There exists a literal l in the body such that $l \in \Delta$.

2. Let the head be p. Then, $p \in \Delta$ or $\overline{p} \in \Delta$ since *deleted_con* for the rule has been invoked.

Let T_1 be a set of rules in $T - \mathcal{R}$ with a literal l such that $l \in \Delta$ or $\overline{l} \in \Delta$ and let T_2 be $T - \mathcal{R} - T_1$. Then, every rule in T_1 satisfies the following conditions.

1. For every literal l in the body, $l \notin \Delta$ and $\overline{l} \notin \Delta$ since otherwise, the rule has been checked.

2. Let the head be p. Then, $p = \bot$ or $p \in \Delta$ since if $p \neq \bot$ and $p \in \Delta$, *rule_con* for the rule has been invoked.

Let M' be a stable model for $\mathcal{R}$. From the assumption, there is no stable model for T which subsumes M'.

This means that there is no stable model for $\mathcal{R}_2 \cup T_2$ which subsumes M' since if there is such a model, every rule in $\mathcal{R}_1 \cup T_1$ is satisfied by the stable model and this contradicts the assumption.

Suppose there is a stable model for T_2 then there is a stable model for $\mathcal{R}_2 \cup T_2$ since there is no common proposition between $\mathcal{R}_2$ and T_2, and $\mathcal{R}_2$ itself has the stable model M'. This contradicts the above result. Therefore, there is no stable model for T_2.

Then, any rule in $T - T_2$ cannot save inconsistency for T_2 even if it is added since any rule in $T \quad T_2$ does not have a proposition in T_2 as its head. Therefore, there is no stable model for T and this contradicts the consistency of T. □.

Proof of Theorem 2: In a similar way to the proof above. □

Proof of Theorem 3: It is sufficient to show the following. □

Lemma 3 *Let T be a logic program and Δ be a set of literals such that it includes $\top$ and for every $l \in \Delta$, $\overline{l} \notin \Delta$.*

1. *Suppose that $derive(p, \Delta)$ terminates for every selection. If there exists a stable model M for $T \cup \mathcal{F}(\Delta)$ such that $M \models p$, then there is a selection of rules for which $derive(p, \Delta)$ succeeds with Δ' and $M \models \mathcal{F}(\Delta')$.*

2. *Suppose that $literal\_con(l, \Delta)$ terminates for every selection. If there exists a stable model M for $T \cup \mathcal{F}(\Delta)$ such that $M \models l$, then there is a selection of rules for which $literal\_con(l, \Delta)$ succeeds with Δ' and $M \models \mathcal{F}(\Delta')$.*

3. *Suppose that $rule\_con(R, \Delta)$ terminates for every selection. If there exists a stable model M for $T \cup \mathcal{F}(\Delta)$ such that $M \models R$, then there is a selection of rules for which $rule\_con(R, \Delta)$ succeeds with Δ' and $M \models \mathcal{F}(\Delta')$.*

4. *Suppose that $deleted\_con(p, \Delta)$ terminates for every selection. If there exists a stable model M for $T \cup \mathcal{F}(\Delta)$, then there is a selection of rules for which $deleted\_con(p, \Delta)$ succeeds with Δ' and $M \models \mathcal{F}(\Delta')$.*

Proof of Lemma: For 1. if there is a stable model which satisfies p, then there is a sequence of supporting rules for p and so, we can always select a rule whose head is p and which is satisfied by M. Others can be proved by induction of number of the longest calls of subprocedures among selections. □

References

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

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

[3] 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).

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

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

[6] Kakas, A. C., Mancarella, P., Stable Theories for Logic Programs, *Proc. of ILPS'91*, pp. 85 – 100 (1991).

[7] Dung, P. M., Negations as Hypotheses: An Abductive Foundation for Logic Programming, *Proc. of ICLP'91*, pp. 3 – 18 (1991).

[8] Loveland, D. W., *Automated Theorem Proving: A Logical Basis*, North-Holland (1978).

[9] Poole, D., Compiling a Default Reasoning System into Prolog, *New Generation Computing*, Vol. 9, No. 1, pp. 3 – 38 (1991).

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

[11] Sadri, F., Kowalski, R., A Theorem-Proving Approach to Database Integrity, *Foundations of Deductive Database and Logic Programming*, (J. Minker, Ed.), Morgan Kaufmann Publishers, pp. 313 – 362 (1988).

[12] Satoh, K., Iwayama, N., Computing Abduction Using the TMS, *Proc. of ICLP'91*, pp. 505 – 518 (1991).