A Query Evaluation Method for Abductive Logic Programming

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

Abstract

We present a query evaluation method for abduction. In artificial intelligence, abduction has been recognized as an important human reasoning applied in various fields [2]. In logic programming, Eshghi and Kowalski [2] introduce abduction to handle negation as failure and Kakas and Mancarella [4, 5] extend the framework to include any arbitrary abducible predicate.

We have already proposed a correct bottom-up procedure to compute abduction [8]. However, this procedure is not suitable for query evaluation. Although [5] proposes a query evaluation method by extending the procedure in [2], there is a problem of incorrectness in the procedure in [2] and this problem is inherited in the procedure in [5]. Also, their procedure can handle only a limited class of integrity constraints.

The procedure we propose in this paper is correct for any consistent abductive framework. If the procedure succeeds, there is a set of hypotheses which satisfies a query, and if the procedure finitely fails, there is no such set. We guarantee correctness since we adopt a forward evaluation of rules and check consistency of "implicit deletion" [7]. Thanks to the forward evaluation of rules, we can also handle any form of integrity constraints.

1 Introduction

In this paper, we present a query evaluation method for abduction in logic programming. Researchers in artificial intelligence have recently recognized that abduction plays an important role in various applications such as diagnosis, planning and design (for example, see [2]).

Eshghi and Kowalski [2] are the first to consider abduction based on stable model semantics [3] in logic programming. They show the relationship between negation as failure and abduction. And they provide a proof procedure to compute negation as failure through abduction for a call-consistent logic program.

This work has been extended intensively by Kakas and Mancarella [4]. They show the relationship of abductive logic programming with autoepstemic logic, assumption-based truth maintenance system and updates, and

also they extend Eshghi and Kowalski's procedure to manipulate arbitrary abducibles [5].

In [8], we also give a bottom-up method of computing a generalized stable model [4] which is a basis of abduction defined by Kakas and Mancarella. Although this method is correct for any general logic program with integrity constraints, it is not goal-directed; that is, even if a goal is given, we might explore some irrelevant parts to a query. Eshghi and Kowalski's top-down procedure [2] solves this problem of irrelevant computation to a query. However, their procedure has a problem of incorrectness for some non-call-consistent programs [2, p.251]. This problem is inherited in the procedure of [5].

Here, we provide a query evaluation method that is correct for every consistent abductive framework [1]. If our procedure answers "yes", then there is a set of hypotheses which satisfies a query. If our procedure answers "no", then there is no set of hypotheses which satisfies a query.

Our procedure can be regarded as an extension of Kakas and Mancarella's procedure [5] in the following two points.

Forward evaluation of rules:
It is important to use integrity constraints to exclude undesirable results from abduction. However, their procedure manipulates a class of integrity constraints in which there is at least one abducible predicate in each integrity constraint.

For example, their procedure cannot handle the following program with an abducible predicate $normal\_bird^{\dagger}$ [2].

$$fly(X) \leftarrow bird(X), normal\_bird^{\dagger}(X) \quad (1)$$
$$bird(tweety) \leftarrow \quad (2)$$
$$non\_fly(tweety) \leftarrow \quad (3)$$
$$\bot \leftarrow fly(X), non\_fly(X) \quad (4)$$

$fly(tweety)$ seems to be derived with an assumption $normal\_bird^{\dagger}(tweety)$ from (1). However, deriving $fly(tweety)$ leads to contradiction by an instance of (4):

$$\bot \leftarrow fly(tweety), non\_fly(tweety).$$

Therefore, $fly(tweety)$ cannot be derived.

This example shows that we need another method to detect contradiction. In our procedure, we use a forward evaluation of rules so that we can check any form of integrity constraints. In the example above, we first derive $fly(tweety)$ from (1), then check the integrity constraint

[1] Consistency of an abductive framework means that there is a generalized stable model for the framework.

[2] Upper-case letters, lower-case letters and $\bot$ express variables, constants and inconsistency, respectively.

$\bot \leftarrow non\_fly(tweety)$ with a forward evaluation of (4) and find contradiction.

Check for Implicit Deletions:
We check consistency for "implicit deletions" first observed by Sadri and Kowalski [7]. For example, consider the following program with an abducible predicate $normal\_barber^{\dagger}$ [3].

$$man(noel) \leftarrow \quad (1)$$
$$barber(noel) \leftarrow \quad (2)$$
$$shaves(noel, X) \leftarrow man(X), \sim shaves(X, X) \quad (3)$$
$$shaves(X, X) \leftarrow barber(X), normal\_barber^{\dagger}(X) \quad (4)$$
$$shaves(casanova, X) \leftarrow barber(X), \sim normal\_barber^{\dagger}(X) \quad (5)$$

$shaves(casanova, noel)$ seems to be derived with an assumption $\sim normal\_barber^{\dagger}(noel)$. However, if we assume it, an instance of (4):

$$shaves(noel, noel) \leftarrow barber(noel), normal\_barber^{\dagger}(noel)$$

is implicitly deleted since its body becomes false. Therefore, contradiction occurs from the instance of (3):

$$shaves(noel, noel) \leftarrow man(noel), \sim shaves(noel, noel).$$

Consequently, $shaves(casanova, noel)$ cannot be derived.

This example shows that we must consider the integrity of rules deleted by the hypothesis. However, the proposed methods [2, 4] do not check consistency of implicit deletion. We believe that this lack of check is a major culprit of incorrectness in these methods. Inversely, we check the implicit deletion in our procedure. So, we can show that if we assume $\sim normal\_barber^{\dagger}(noel)$, neither $shaves(noel, noel)$ nor $\sim shaves(noel, noel)$ can be consistently derived in the example above.

Also, our procedure is important in the following respects.

1. If we do not consider abducible predicates, then our method can be used for query evaluation of every consistent general logic program with integrity constraints based on stable model semantics.

2. Our procedure adopts integrity checking for addition of a rule by accumulating hypotheses. In some cases, it avoids infinite loops which occur in other methods of integrity checking such as [7].

The structure of the paper is as follows. Firstly, we review the definitions of an abductive framework. Then, we show the procedure for query evaluation of an abductive framework and give some examples. Finally, we compare our method with related researches. For proofs of the theorems, see [9].

[3] $\sim$ expresses negation as failure.

2 A Semantics of Abductive Framework

We mainly follow the definition of abductive framework in [4], but we modify it slightly for notational conveniences. Firstly, we define a rule and an integrity constraint.

Definition 2.1 *Let H be an atom, and $L_1, ..., L_m (m \geq 0)$ be literals each of which is an atom or a negated atom of the form $\sim B$. A rule is of the form:*

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

We call H the *head* of the rule and $L_1, ..., L_m$ the *body* of the rule. Let R be a rule. $head(R)$, $body(R)$ and $pos(R)$ denote the head of R, the set of literals in the body of R and the set of positive literals in $body(R)$ respectively.

Definition 2.2 *Let $L_1, ..., L_m (m \geq 0)$ be literals. An integrity constraint is of the form:*

$$\bot \leftarrow L_1, L_2, ..., L_m \ ^4.$$

We write integrity constraints as the above form so that we do not have to distinguish integrity constraints and rules. So, from this point, we do not distinguish rules and integrity constraints.

Moreover, we impose that rules in a program must be *range-restricted*, that is, any variable in a rule R must occur in $pos(R)$. However, [1] pointed out that any rule can be translated into range-restricted form by inserting a new predicate "*dom*" describing Herbrand universe for every non-range-restricted variables in the rule.

For a given program (with integrity constraints), we define a stable model as follows.

Definition 2.3 *Let T be a logic program and Π_T be a set of ground rules obtained by replacing all variables in each rule in T by every element of its Herbrand universe. Let M be a set of ground atoms from Π_T and Π_T^M be the following (possibly infinite) program.*

$$\Pi_T^M = \{H \leftarrow B_1, ..., B_k \mid H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m \in \Pi_T \text{ and } A_i \notin M \text{ for each } i = 1, ..., m.\}$$

Let $min(\Pi_T^M)$ be the least model of Π_T^M. A stable model *for a logic program T is M iff $M = min(\Pi_T^M)$ and $\bot \notin M$.*

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.

For a query evaluation procedure, it is better to limit ground instances which should be considered. For example, consider the following program.

[4] Although we only consider the above form of integrity constraints (denials) in this paper, there is a transformation from a more general form of integrity constraints to denials as shown in [7].

$p(1,2) \leftarrow p(2,1)$ (1)
$p(2,1) \leftarrow \sim q(2)$ (2)
$q(X) \leftarrow p(X,Y), \sim q(Y)$ (3)
$r(f(1)) \leftarrow$ (4)

From the above definition, we have to consider infinite ground rules for (3) because of the function symbol f in (4). However, it is clear that there is no possibility to make any instances for $p(X,Y)$ other than $p(1,2)$ and $p(2,1)$ to be true, and so, all we have to consider are actually the following two ground rules.

$q(1) \leftarrow p(1,2), \sim q(2)$ (3.1)
$q(2) \leftarrow p(2,1), \sim q(1)$ (3.2)

We formalize this phenomenon as follows.

Definition 2.4 *Let T be a logic program and T^- be a negation-removed program obtained by removing all integrity constraints in T and all the negative literals in the body of remaining rule and $min(T^-)$ be the least minimal model of T^-. We define a* relevant ground program Ω_T *for T as follows:*

$\Omega_T = \{H \leftarrow B_1, ..., B_k, \sim A_1, ..., \sim A_m \in \Pi_T | B_i \in min(T^-)$ *for each* $i = 1, ..., k.\}$

Proposition 2.5 *Let T be a logic program. A set of stable models for T is equal to a set of stable models for Ω_T.*

The above proposition actually holds for any logic program, but if we impose a program to be range-restricted, we can construct Ω_T directly from T^- without considering instantiation of variables with every elements in Herbrand Universe.

Example 2.6 *Let T be the above program (1), (2), (3) and (4).*
Then, $min(T^-)$ is $\{p(1,2), p(2,1), q(1), q(2), r(f(1))\}$ and so, Ω_T becomes (1), (2), (3.1), (3.2) and (4).
Ω_T has only one stable model which is equal to the unique stable model for T, that is, $\{p(1,2), p(2,1), q(1), r(f(1))\}$.

Now, we define an *abductive framework.*

Definition 2.7 *An* abductive framework *is a pair $\langle T, A \rangle$ where A is a set of predicate symbols, called* abducible predicates *and T is a set of rules each of whose head is not in A.*

We call a set of all ground atoms for predicates in A *abducibles.* As pointed out in [4], we can translate a program which includes a definition of abducibles to an equivalent framework that satisfies the above requirement. Moreover, we impose an abductive framework to be range-restricted, that is, any variable in a rule of a program must occur in non-abducible positive literals of the rule.

Now, we define a semantics of an abductive framework.

Definition 2.8 *Let $\langle T, A\rangle$ be an abductive framework and Θ be a set of abducibles. A* generalized stable model *$M(\Theta)$ is a stable model of $T \cup \{H \leftarrow | H \in \Theta\}$.*

We say that $\langle T, A\rangle$ is *consistent* if there exists a generalized stable model $M(\Theta)$ for some Θ. The similar proposition to Proposition 2.5 holds for an abductive framework.

Proposition 2.9 *Let $\langle T, A\rangle$ be a logic program and T^- be an abducible-and-negation-removed program obtained by removing all negative literals and abducibles in the body of each rule in a program T. We define a* relevant ground program *Ω_T for $\langle T, A\rangle$ as follows:*

$\Omega_T = \{H \leftarrow B_1, ..., B_k, C_1, ..., C_l, \sim A_1, ..., \sim A_m \in \Pi_T |$
$B_i \in min(T^-)$ *for each* $i = 1, ..., k$ *and* $C_1, ..., C_l$ *are abducibles.*$\}$

Then, a set of generalized stable models for $\langle T, A\rangle$ is equal to a set of generalized stable models for $\langle \Omega_T, A\rangle$.

3 Query Evaluation for Abduction

Before showing our query evaluation method, we need the following definitions. Let l be a literal. Then, $\overline{l}$ denotes the complement of l.

Definition 3.1 *Let T be a logic program. A set of resolvents w.r.t. a ground literal l and T, $resolve(l, T)$ is the following set of rules:*

$resolve(l, T) =$
$\{(\bot \leftarrow L_1, ..., L_k)\theta |$ *l is negative and*
$H \leftarrow L_1, ..., L_k \in T$ *and* $\overline{l} = H\theta$ *by a ground substitution* $\theta\} \cup$
$\{(H \leftarrow L_1, ..., L_{i-1}, L_{i+1}, ..., L_k)\theta |$
$H \leftarrow L_1, ..., L_k \in T$ *and* $l = L_i\theta$ *by a ground substitution* $\theta\}$

The first set of resolvents are for negation as failure and the second set of resolvents corresponds with "forward" evaluation of the rule introduced in [7].

Example 3.2 *Consider the following program T.*

$p(1, 2) \leftarrow p(2, 1)$ *(1)*
$p(2, 1) \leftarrow \sim q(2)$ *(2)*
$q(X) \leftarrow p(X, Y), \sim q(Y)$ *(3)*

Then, $resolve(\sim q(2), T)$ is a set of the following rules:

$p(2, 1) \leftarrow$ *with the literal in the body of (2)*
$\bot \leftarrow p(2, Y), \sim q(Y)$ *with the head of (3)*
$q(X) \leftarrow p(X, 2)$ *with the second literal in the body of (3)*

Definition 3.3 *Let T be a logic program. A set of deleted rules w.r.t. a ground literal l and T, $del(l,T)$, is the following set of rules:*

$$del(l,T) = \{(H \leftarrow L_1,...,L_k)\theta | \\ H \leftarrow L_1,...,L_k \in T \text{ and } \overline{l} = L_i\theta \text{ by a ground substitution } \theta\}$$

Example 3.4 *Consider the program T in Example 3.2. Then, $del(q(2),T)$ is a set of the following rules:*

$p(2,1) \leftarrow \sim q(2)$ *from (2)*
$q(X) \leftarrow p(X,2), \sim q(2)$ *from (3)*

Our query evaluation procedure consists of 4 subprocedures, $derive(p,\Delta)$, $literal\_con(l,\Delta)$, $rule\_con(R,\Delta)$ and $deleted\_con(R,\Delta)$ where p is a non-abducible atom and Δ is a set of ground literals already assumed and l is a ground literal and R is a rule.

$derive(p,\Delta)$ returns a ground substitution for the variables in p and a set of ground literals. This set of ground literals is a union of Δ and literals newly assumed during execution of the subprocedure. Other subprocedures return a set of ground literals.

The subprocedures have a **select** operation and a **fail** operation. The **select** operation expresses a nondeterministic choice among alternatives. The **fail** operation expresses immediate termination of an execution with failure. Therefore, a subprocedure succeeds when its inner calls of subprocedures do not encounter **fail**. We say *a subprocedure succeeds with (θ and) Δ* when the subprocedure successfully returns (θ and) Δ.

The informal specification of the 4 subprocedures is as follows.

1. $derive(p,\Delta)$ searches a rule R of p in a program T whose body can be made true with a ground substitution θ under a set of assumptions Δ. To show that every literal in the body can be made true, we call *derive* for non-abducible positive literals in the body. Then, we check the consistency of other literals in the body with T and Δ. Note that because of the range-restrictedness, other literals in R become ground after all the calls of *derive* for non-abducible positive literals.

2. $literal\_con(l,\Delta)$ checks the consistency of a ground literal l with T and Δ. To show the consistency for assuming l, we add l to Δ; then, we check the consistency of resolvents and deleted rules w.r.t. l and T.

3. $rule\_con(R,\Delta)$ checks the consistency of a rule R with T and Δ. If R is not ground, we must check the consistency for ground instances of R. But by Proposition 2.9, it is sufficient to consider every ground instance $R\theta$ in $\Omega_{T\cup\{R\}}$. We can prove the consistency by showing that either a literal in $body(R\theta)$ can be falsified or $body(R\theta)$ can be made true and $head(R\theta)$ consistent.

   This procedure can also be used to check integrity for rule addition.

4. *deleted_con*(R, Δ) checks if a deletion of R does not cause any contradictions with T and Δ. To show the consistency of the implicit deletion of R, it is sufficient to prove that the head of every ground instance $R\theta$ in Ω_T [5] can be made either true or false.

Thanks to the range-restrictedness, we can compute all ground instances of a rule R (if they are finite) in Ω_T (or $\Omega_{T\cup\{R\}}$). For this, we compute every SLD derivation of a query which consists of all non-abducible positive literals in $body(R)$ to the abducible-and-negation-removed program T^- (or $(T \cup \{R\})^-$).

Example 3.5 *Consider T and the second resolvent of $resolve(\sim q(2), T)$ in Example 3.2. Let R be the second resolvent and T_1 be $T \cup \{R\}$. In order to obtain possible ground instances of R in Ω_{T_1}, we compute every SLD derivation of a query $? - p(2, Y)$ to the program T_1^-. Then, since only the returned substitution from SLD derivation is $\{Y/1\}$, we have the following ground instance in Ω_{T_1} for R:*

$$\bot \leftarrow p(2,1), \sim q(1)$$

Now, we describe in detail the subprocedures in Figure 1 and Figure 2. In Figure 1, ε denotes empty substitution and $\theta_i\sigma_i$ expresses a composition of two substitutions θ_i and σ_i. Also, we denote a set of non-abducible positive literals, non-abducible negative literals, and abducibles (either negative or positive) in a rule R as $pos(R)$, $neg(R)$ and $abd(R)$.

If we remove *deleted_con* and do not consider resolvents obtained with "forward" evaluation of the rule, then this procedure coincides with that of Kakas and Mancarella [4]. That is, our procedure is obtained by augmenting their procedure with an integrity constraint checking in a bottom-up manner and with an implicit deletion checking.

We can show the following theorems for correctness of successful derivation and finite failure.

Theorem 3.6 *Let $\langle T, A\rangle$ be a consistent abductive framework. Suppose $derive(p, \{\})$ succeeds with (θ, Δ). Then, there exists a generalized stable model $M(\Theta)$ for T such that Θ includes all positive abducibles in Δ and $M(\Theta) \models p\theta$.*

This theorem means that if the procedure $derive(p, \{\})$ answers "yes" with (θ, Δ), then there is a generalized stable model which satisfies $p\theta$. However, we cannot say in general that we make $p\theta$ true only with positive abducibles in Δ, because there might be some hypotheses which are irrelevant to a query but which we must assume to get consistency.

The following is a theorem related to correctness for finite failure.

[5] Note that $\Omega_{T\cup\{R\}} = \Omega_T$ since R is an instance of a rule in T.

```
derive(p, Δ) p: a non-abducible atom; Δ: a set of literals
begin
  if p is ground and p ∈ Δ then return (ε, Δ)
  elseif p is ground and ∼p ∈ Δ then fail
  else
  begin
    select R ∈ T s.t. head(R) and p are unifiable with an mgu θ
    if such a rule is not found then fail
    Δ_0 := Δ, θ_0 := θ, B_0 := pos(Rθ), i := 0
    while B_i ≠ {} do
    begin
      take a literal l in B_i
      if derive(l, Δ_i) succeeds with (σ_i, Δ_{i+1})
        then θ_{i+1} := θ_i σ_i, B_{i+1} := (B_i − {l})σ_i, i := i + 1 and continue
    end
    δ := θ_i
    for every l ∈ neg(Rδ) ∪ abd(Rδ) do
    begin
      if literal_con(l, Δ_i) succeeds with Δ_{i+1}
        then i := i + 1 and continue
    end
    if literal_con(pδ, Δ_i) succeeds with Δ' then return (δ, Δ')
  end
end (derive)

literal_con(l, Δ) l: a ground literal; Δ: a set of literals
begin
  if l ∈ Δ then return Δ
  elseif l = ⊥ or 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 R ∈ del(l, T) do
      if deleted_con(R, Δ_i) succeeds with Δ_{i+1}
        then i := i + 1 and continue
  end
  return Δ_i
end (literal_con)
```

Figure 1: The definition of *derive* and *literal_con*

$rule\_con(R, \Delta)$ R: a rule; Δ: a set of literals
begin
  $\Delta_0 := \Delta$, $i := 0$
  for every ground rule $R\theta \in \Omega_{T \cup \{R\}}$ **do**
  begin
    select (a) or (b)
    (a) **select** $l \in body(R\theta)$
      if $l \in pos(R\theta) \cup abd(R\theta)$ and $literal\_con(\bar{l}, \Delta_i)$ succeeds with Δ_{i+1}
        then $i := i + 1$ and **continue**
      elseif $l \in neg(R\theta)$ and $derive(\bar{l}, \Delta)$ succeeds with $(\varepsilon, \Delta_{i+1})$
        then $i := i + 1$ and **continue**
    (b) $\Delta_i^0 := \Delta_i$, $j := 0$
      for every $l \in body(R\theta)$ **do**
      begin
        if $l \in pos(R\theta)$
          and $derive(l, \Delta_i^j)$ succeeds with $(\varepsilon, \Delta_i^{j+1})$
          then $j := j + 1$ and **continue**
        elseif $l \in neg(R\theta) \cup abd(R\theta)$
          and $literal\_con(l, \Delta_i^j)$ succeeds with Δ_i^{j+1}
          then $j := j + 1$ and **continue**
      end
      if $literal\_con(head(R\theta), \Delta_i^j)$ succeeds with Δ_{i+1}
        then $i := i + 1$ and **continue**
  end
  return Δ_i
end $(rule\_con)$

$deleted\_con(R, \Delta)$ R: a rule; Δ: a set of literals
begin
  $\Delta_0 := \Delta$, $i := 0$
  for every ground rule $R\theta \in \Omega_T$ **do**
  begin
    select (a) or (b)
    (a) **if** $derive(head(R\theta), \Delta_i)$ succeeds with $(\varepsilon, \Delta_{i+1})$
      then $i := i + 1$ and **continue**
    (b) **if** $literal\_con(\sim head(R\theta), \Delta_i)$ succeeds with Δ_{i+1}
      then $i := i + 1$ and **continue**
  end
  return Δ_i
end $(deleted\_con)$

Figure 2: The definition of *rule_con* and *deleted_con*

Theorem 3.7 *Let $\langle T, A\rangle$ be an abductive framework. Suppose that every selection of rules terminates for $derive(p, \{\})$ with either success or failure. If there exists a generalized stable model $M(\Theta)$ for $\langle T, A\rangle$ and a ground substitution θ such that $M(\Theta) \models p\theta$, then there is a selection of rules such that $derive(p, \{\})$ succeeds with (θ, Δ) where Θ includes all positive abducibles in Δ.*

This theorem means that if we can search exhaustively in selecting the rules and there is a generalized stable model which satisfies a query, then the procedure always answers "yes".

With this theorem, we obtain the following corollary for a finite failure.

Corollary 3.8 *Let $\langle T, A\rangle$ be an abductive framework. If $derive(p, \{\})$ fails, then for every generalized stable model $M(\Theta)$ for $\langle T, A\rangle$ and for every ground substitution θ, $M(\Theta) \not\models p\theta$.*

When the procedure $derive(p, \{\})$ answers "no", there is no generalized stable model which satisfies the query. Also, this corollary means that we can use a finite failure to check if the negation of a ground literal is true in all generalized stable models since finite failure of $derive(p, \{\})$ means that every generalized stable model satisfies $\sim p$.

4 Examples

Example 4.1 *Consider the program T in Example 3.2 and an abductive framework $\langle T, \emptyset\rangle$. Then, Figure 3 shows a sequence of calling procedures obtained for $derive(q(V), \{\})$.*

In Figure 3, we firstly search a rule whose head is unifiable with $q(V)$ (Step 2) and try to make the body of the rule to be true (Step 3$\sim$20). There are two literals in the body, $p(V, Y_1)$ and $\sim q(Y_1)$. We find a ground substitution for $p(V, Y_1)$ (Step 3$\sim$19) and then show consistency of $\sim q(2)$ (Step 20).

While finding a substitution for the first literal $p(V, Y_1)$, we check consistency of $\sim q(2)$ (Step 7$\sim$17). To show its consistency, we check three resolvents for $\sim q(2)$ shown in Example 3.2 (Step 8, 16 and 17).

During the check for the first resolvent, we check implicit deletion of an instance of rule (3) deleted by $q(1)$ (Step 14) in order to show consistency of $q(1)$ (Step 13). In this case, we have a non-ground deleted rule and so, we compute every ground instance in a relevant ground program. There is only one such ground rule $(q(2) \leftarrow p(2,1), \sim q(1))$ and since $\sim q(2)$ has been already assumed, Step 14 succeeds.

Similarly, at Step 16 and 17, we have non-ground rules and so, we compute ground instances in a relevant ground program. Such instances are $(\bot \leftarrow p(2,1), \sim q(1))$ for Step 16 and $(q(1) \leftarrow p(1,2))$ for Step 17. Both steps succeed since $q(1)$ and $p(1,2)$ has been already assumed.

$derive(q(V), \{\})$ 1
  select $q(V) \leftarrow p(V, Y_1), \sim q(Y_1)$ 2
    $derive(p(V, Y_1), \{\})$ 3
      select $p(1,2) \leftarrow p(2,1)$ 4
        $derive(p(2,1), \{\})$ 5
          select $p(2,1) \leftarrow \sim q(2)$ 6
            $lit\_con(\sim q(2), \{\})$ 7
              $rule\_con((p(2,1) \leftarrow), \{\sim q(2)\})$ 8
                $lit\_con(p(2,1), \{\sim q(2)\})$ 9
                  $rule\_con((p(1,2) \leftarrow), \{p(2,1), \sim q(2)\})$ 10
                    $lit\_con(p(1,2), \{p(2,1), \sim q(2)\})$ 11
                      $rule\_con((q(1) \leftarrow \sim q(2)), \{p(1,2), p(2,1), \sim q(2)\})$ 12
                        $lit\_con(q(1), \{p(1,2), p(2,1), \sim q(2)\})$ 13
                          $del\_con((q(X_2) \leftarrow p(X_2,1), \sim q(1)),$
                            $\{q(1), p(1,2), p(2,1), \sim q(2)\})$ 14
                  $rule\_con((q(2) \leftarrow \sim q(1)), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 15
              $rule\_con((\bot \leftarrow p(2, Y_3), \sim q(Y_3)), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 16
              $rule\_con((q(X_4) \leftarrow p(X_4, 2)), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 17
            $lit\_con(p(2,1), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 18
        $lit\_con(p(1,2), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 19
    $lit\_con(\sim q(2), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 20
    $lit\_con(q(1), \{q(1), p(1,2), p(2,1), \sim q(2)\})$ 21

ANSWER
$V = 1$ under $\{q(1), p(1,2), p(2,1), \sim q(2)\}$

Figure 3: Calling Sequence for $derive(q(V), \{\})$

Finally, we check consistency of $q(1)$ (Step 21) and get a ground substitution of $\{V/1\}$ for $q(V)$.

Example 4.2 *Consider the following program T and an abductive framework $\langle T, \{normal\_bird^{\dagger}\}\rangle$.*

$fly(X) \leftarrow bird(X), normal\_bird^{\dagger}(X)$ *(1)*
$bird(tweety) \leftarrow$ *(2)*
$non\_fly(tweety) \leftarrow$ *(3)*
$\bot \leftarrow fly(X), non\_fly(X)$ *(4)*

Then, Figure 4 shows a sequence of calling procedures obtained for $derive(fly(tweety), \{\})$ by left-most depth-first search [6].

In Figure 4, we firstly try to search the rule for $f(t)$ and find the rule (1) (Step 2) and try to make the body true. Then, we check consistency for one of the literal $b(t)$ in the body (Step 5$\sim$15) and assume $\sim nb^{\dagger}(t)$ at Step

[6] In Figure 4, f, b, $nb^{\dagger}$, nf and t mean fly, $bird$, $normal\_bird^{\dagger}$, non_fly and $tweety$ respectively.

$derive(f(t), \{\})$ 1
  select $f(t) \leftarrow b(t), nb^{\dagger}(t)$ 2
    $derive(b(t), \{\})$ 3
      select $b(t) \leftarrow$ 4
        $lit\_con(b(t), \{\})$ 5
          $rule\_con((f(t) \leftarrow nb^{\dagger}(t)), \{b(t)\})$ 6
            $lit\_con(\sim nb^{\dagger}(t), \{b(t)\})$ 7
              $del\_con((f(t) \leftarrow b(t), nb^{\dagger}(t)), \{\sim nb^{\dagger}(t), b(t)\})$ 8
                $derive(f(t), \{\sim nb^{\dagger}(t), b(t)\})$ 9
                  select $f(t) \leftarrow b(t), nb^{\dagger}(t)$ 10
                    $derive(b(t), \{\sim nb^{\dagger}(t), b(t)\})$ 11
                    $lit\_con(nb^{\dagger}(t), \{\sim nb^{\dagger}(t), b(t)\}) \Longrightarrow$**fail** (back to 8) 12
                $lit\_con(\sim f(t), \{\sim nb^{\dagger}(t), b(t)\})$ 13
                  $rule\_con((\bot \leftarrow b(t), nb^{\dagger}(t)), \{\sim f(t), \sim nb^{\dagger}(t), b(t)\})$ 14
                  $del\_con((\bot \leftarrow f(t), nf(t)), \{\sim f(t), \sim nb^{\dagger}(t), b(t)\})$ 15
    $lit\_con(nb^{\dagger}(t), \{\sim f(t), \sim nb^{\dagger}(t), b(t)\}) \Longrightarrow$**fail** (back to 6) 16
          $lit\_con(nb^{\dagger}(t), \{b(t)\})$ 17
            $rule\_con((f(t) \leftarrow b(t)), \{nb^{\dagger}(t), b(t)\})$ 18
              $lit\_con(f(t), \{nb^{\dagger}(t), b(t)\})$ 19
                $rule\_con((\bot \leftarrow nf(t)), \{f(t), nb^{\dagger}(t), b(t)\})$ 20
                  $lit\_con(\sim nf(t), \{f(t), nb^{\dagger}(t), b(t)\})$ 21
                    $rule\_con((\bot \leftarrow),$
                      $\{\sim nf(t), f(t), nb^{\dagger}(t), b(t)\}) \Longrightarrow$**fail**(back to 20) 22
                  $derive(nf(t), \{f(t), nb^{\dagger}(t), b(t)\})$ 23
                    select $nf(t) \leftarrow$ 24
                      $lit\_con(nf(t), \{f(t), nb^{\dagger}(t), b(t)\})$ 25
                        $rule\_con((\bot \leftarrow f(t)),$
                          $\{nf(t), f(t), nb^{\dagger}(t), b(t)\}) \Longrightarrow$**fail** 26

Figure 4: Calling Sequence for $derive(fly(tweety), \{\})$

7. After showing consistency of $\sim nb^{\dagger}(t)$ (Step 7$\sim$15), we try to make the other literal $nb^{\dagger}(t)$ in the body of the selected rule at Step 2, but we fail (Step 16). Then, we backtrack and assume $nb^{\dagger}(t)$ instead (Step 17). Since $nb^{\dagger}$ is an abducible predicate, it is sufficient to show consistency of $nb^{\dagger}(t)$. However, to show its consistency, we must show consistency of $f(t)$ (Step 19). Unfortunately, contradiction occurs by the integrity constraint (4) (Step 20$\sim$26) and therefore, we cannot conclude $f(t)$.

5 Related Work

5.1 The procedure of Kakas and Mancarella

As stated in Section 3, if we do not check the consistency for implicit deletion and we do not consider the "forward" evaluation of rules, our procedure is

identical to that of Kakas and Mancarella [5]. Although "forward" evaluation of rules is used mainly to forward-check the integrity constraint, it is also necessary for implicit deletion check. This is because a rule might be deleted by both an assumed literal itself; and also by other literals derived from the assumed literal. Therefore, the whole procedure is necessary for consistency checking of an abductive framework; and also of a general logic program even *without* integrity constraints.

5.2 Integrity check method of Sadri and Kowalski

Sadri and Kowalski [7] propose an integrity check method by augmenting the SLDNF procedure with "forward" evaluation of rules and consistency check for implicit deletion. Although we use also the same techniques, our method differs from theirs in accumulating hypotheses during integrity check. The technique of hypothesis accumulation enables us to prove the consistency for addition of rules for a wider class of logic programs more than with their method. For example, consider the following program:

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

To check the consistency for addition of p, they invoke a query $\leftarrow p$ and see if it finitely fails. However, their procedure enters an infinite loop, whereas $rule\_con((p \leftarrow), \{\})$ of our procedure succeeds in showing the consistency for addition because of the accumulation of hypotheses.

Moreover, their method guarantees consistency for addition of a rule not for every general logic program, but for a limited class of logic programs which contain no negative literals in the body of each rule. On the other hand, if $rule\_con(R, \{\})$ succeeds, we can guarantee that R is consistent with the current program even if it contains negative literals in its body.

5.3 Poole's Theorist

Poole [6] develops a default and abductive reasoning system called *Theorist*. Our method differs from *Theorist* in the following points.

1. The basic language for *Theorist* is a first-order language whereas we use a logic program. So, in *Theorist*, a contrapositive inference must be considered, while in our setting, it is not necessary. Instead of that, however, we must consider the consistency checking of implicit deletion.

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 deals with a default theory with only arbitrary default rules and no proper axioms.

6 Conclusion

In this paper, we propose a query evaluation method for an abductive framework. Our procedure can be regarded as an extension of the procedure of Kakas and Mancarella by adding forward evaluation of rules and consistency check for implicit deletion.

We think that we need to investigate the following research in the future.

1. In our method, a literal l for $literal\_con(l, \Delta)$ must be ground. This restriction imposes a program to be range-restricted. However, if we can manipulate non-ground hypotheses, range-restrictedness is no longer necessary. Therefore, we would like to investigate a direct treatment of non-ground hypotheses.

2. We should investigate the computational complexity of our procedure and compare it with our bottom-up procedure for abduction [8].

Acknowledgments

We thank Katsumi Inoue from ICOT, Bob Kowalski from Imperial College, Tony Kakas from University of Cyprus, Phan Minh Dung from AIT, Chris Preist from HP Labs. and anonymous referees for instructive comments.

References

[1] Manthey, R., Bry, F., SATCHMO: A Theorem Prover Implemented in Prolog, *Proc. of CADE'88*, pp. 415 – 434 (1988).

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

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

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

[5] Kakas, A. C., Mancarella, P., On the Relation between Truth Maintenance and Abduction, *Proc. of PRICAI'90*, pp. 438 – 443 (1990).

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

[7] 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).

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

[9] Satoh, K., Iwayama, N., A Query Evaluation Method for Abductive Logic Programming ICOT Technical Report, ICOT(1992).