ICOT Technical Report: TR-0861

TR-0861

Computing Abduction in Logic Programming

by

N. Iwayama & K. Satoh

論文

Computing Abduction in Logic Programming

Noboru Iwayama
Ken Satoh
Fujitsu Laboratories Ltd., 1015 Kamikodanaka, Kawasaki 211, Japan

Keywords: abduction, logic programming, generalized stable model, integrity constraint.

Summary

We present a method to compute abduction discussed in [Eshghi89, Kakas 90a, Kakas 90b]. We translate an abductive framework into a normal logic program with integrity constraints and show a correspondence between generalized stable models defined in [Kakas 90a, Kakas 90b] and stable models for the translation of the abductive framework. Based on the correspondence, we can find abductive explanations for an observation from the stable models for the translated program with a special kind of integrity constraint for the observation.

Then, we show a bottom-up procedure to compute stable models for a normal logic program with integrity constraints. The proposed procedure can exclude the unnecessary construction of stable models on early stages of the procedure by checking integrity constraints during the construction and deriving some facts from integrity constrains. In general, a bottom-up procedure has disadvantage of constructing stable models not related to an observation for computing abductive explanations. However, our procedure avoids the disadvantage by expecting which rule should be used for satisfaction of an integrity constraint for the observation and starting bottom-up computation based on the expectation.

1 Introduction

We present a method of calculating abduction discussed in [Eshghi89, Kakas 90a, Kakas 90b]. Recent researches have revealed that abduction plays an important role in artificial intelligence, as stated in [Eshghi89]. Various researchers have studied abduction in terms of logic programming framework [Eshghi89, Kakas 90a, Kakas 90b] (called *abductive logic programming*) and shown relationships with nonmonotonic reasoning framework such as negation as failure, assumption-based truth maintenance system and autoepistemic logic.

In this paper, we give a method of computing abduction in logic programming. To do that, we first give a translation of an abductive framework to a normal logic program with integrity constraints and show that a stable model [Gelfond 88] for the translated logic program coincides with some generalized stable model for the abductive framework defined in [Kakas 90b]. Then, we provide a nondeterministic bottom-up procedure which calculates stable models for a normal logic program with integrity constraints. With the above translation, our procedure computes abduction.

Kakas and Mancarella [Kakas 90a] have provided an abductive proof procedure to calculate an explanation for a given observation in abductive logic programming. They extend Eshghi's top-down procedure for abduction [Eshghi89] in order to manipulate arbitrary hypotheses. However, their procedure inherits a problem of Eshghi's procedure that correctness does not hold in general for logic programs with recursion as shown in [Eshghi89, p.251]. In contrast to their procedure, our method is correct for any abductive logic program because of correctness of our procedure.

One may suspect that a bottom-up procedure is not efficient for computing abductive explanations for an observation. We can compute hypotheses for an explanation by computing generalized stable models and then taking out hypotheses from the generalized stable models satisfying the observation. This is apparently inefficient, because many of generalized stable models not related to the observation might be constructed.

We can overcome the inefficiency of this naive method as follows. We show that a set of hypotheses used for explanation defined in abductive framework coincides with a part of the stable models for the translated logic program which satisfy a special integrity constraint representing an observation. According to the coincidence, we enhance our procedure to use information given in the observation by top-down expectation for the integrity constraint added for the observation. The idea of this enhancement is to search a rule which has the possibility of satisfying the integrity constraint and if such a rule does not exist, the procedure immediately fails.

Moreover, we enhance our procedure to save search space as much as possible by excluding unsatisfiable stable models by checking integrity constraints during computing stable models dynamically and deriving some facts actively from integrity constraints.

The structure of the paper is as follows. In section 2, we show a translation of an abductive framework to a normal logic program with integrity constraints. In section 3, we provide a procedure which calculates a stable model for a normal logic program with integrity constraints. In section 4, we show examples of computation by the procedure. In the last two sections, we note related works and conclude the paper. A preliminary version of this paper is presented in [Satoh 91].

2 Translating Abductive Framework to Logic Program with Integrity Constraints

We follow the definition in [Kakas 90a, Kakas 90b] but restrict ourselves to considering 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.

Firstly, we define a normal logic program and integrity constraints.

Definition 1 *Let A_i be propositional symbols. A* normal logic program *consists of (possibly countably infinite) rules of the form:*

$$A_0 \leftarrow A_1, ..., A_m, not A_{m+1}, ..., not A_n.$$

We call A_0 the *head* of the rule and $A_1, ..., A_n$ the *body* of the rule. Let R be a rule. We denote the head of R as $head(R)$ and the set of propositions $A_1, ..., A_m$ in R as $pos(R)$ and the set of propositions $A_{m+1}, ..., A_n$ in R as $neg(R)$.

Definition 2 *Let A_i be propositions. A set of* integrity constraints *consists of (possibly countably infinite) formulas of the form:*

$$\leftarrow A_1, ..., A_m, not A_{m+1}, ..., not A_n.$$

Let C be an integrity constraint. We denote the head of R as $head(C)$ and the set of propositions $A_1, ..., A_m$ in C as $pos(C)$ and the set of propositions $A_{m+1}, ..., A_n$ in R as $neg(C)$.

We extend the definition of stable models in [Gelfond 88] for a normal logic program with integrity constraints as follows.

Definition 3 *A pair $\langle T, I \rangle$ be a normal logic program with integrity constraints, where T is a normal logic program and I is a set of integrity constraints. A* stable model *for a normal logic program with integrity constraints 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 if some $n \in neg(R)$ is in M*
   (b) *Deleting every negated atom in the remaining 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 this as $M \models C$.*

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

We follow the definition of abductive framework in [Kakas 90a, Kakas 90b].

Definition 4 *An* abductive framework *is a triple $\langle T, A, I \rangle$ where*

1. *A is a (possibly countably infinite) set of propositional symbols called* abducible propositions.

2. *T is a normal logic program where no rule's head is equal to any element of A.*

3. *I is a set of integrity constraints*[1].

We follow the definition of generalized stable models and explanation with respect to $\langle T, A, I\rangle$ in [Kakas 90a, Kakas 90b].

Definition 5 *Let $\langle T, A, I\rangle$ be an abductive framework and Δ be a subset of A. A* generalized stable model *$M(\Delta)$ of $\langle T, A, I\rangle$ is a stable model of $\langle T \cup \mathcal{F}(\Delta), I\rangle$ where $\mathcal{F}(\Delta) = \{p \leftarrow \mid p \in \Delta\}$.*

Definition 6 *Let $\langle T, A, I\rangle$ be an abductive framework, and q be a proposition called* observation, *and Δ be a subset of A. q has an* abductive explanation *with a set of hypotheses Δ if and only if there exists a generalized stable model $M(\Delta)$ such that $q \in M(\Delta)$.*

At first sight, an abductive framework seems to extend logic programming by introducing abducibles, but it turns out that we can embed an abductive framework into a logic program with integrity constraints. We translate an abductive framework as follows.

Translation of abductive framework

Let $\langle T, A, I\rangle$ be an abductive framework.

1. For each abducible p in A, we introduce a new proposition $\bar{p}$ which is not used in $\langle T, A, I\rangle$.

[1]To be precise, we only consider a special form of integrity constraints whereas in [Kakas 90a, Kakas 90b] they allow any form of integrity constraints. However, those constraints can be translated into our form of integrity constraints.

2. We add the following pair of rules in T for each abducible p in A:

$$p \leftarrow not\, \tilde{p} \text{ and } \tilde{p} \leftarrow not\, p.$$

We denote the set of added rules as $\Gamma(A)$, that is:

$$\Gamma(A) = \{p \leftarrow not\, \tilde{p} | p \in A\} \cup \{\tilde{p} \leftarrow not\, p | p \in A\}$$

3. We obtain a normal logic program with integrity constraints $\langle T \cup \Gamma(A), I\rangle$ as the translation of the abductive framework $\langle T, A, I\rangle$.

The above pair of rules expresses that p and $\tilde{p}$ are mutually exclusive. So, $\tilde{p}$ intuitively means that p is not believed. The first rule"$p \leftarrow not\, \tilde{p}$" is used to assume p and the second rule "$\tilde{p} \leftarrow not\, p$" is used not to assume p. Especially, if we get a contradiction by assuming p, the latter rule is used to prevent the former rule from being used to assume p.

Then we have the following correspondence between abductive framework and its translation.

Theorem 1 *Let $\langle T, A, I\rangle$ be an abductive framework and $\langle T \cup \Gamma(A), I\rangle$ be its translation and Δ be a subset of A. $M(\Delta)$ is a generalized stable model of $\langle T, A, I\rangle$ if and only if there exists a stable model M' for $\langle T \cup \Gamma(A), I\rangle$ such that $M' = M(\Delta) \cup \tilde{\nabla}$ where $\tilde{\nabla} = \{\tilde{p} | p \in (A - \Delta)\}$.*

Proof: See Appendix.

Example 1 *Generalized stable models [Kakas 90b, p.387]*

Consider the following logic program T:

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

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

with abducibles $A = \{a, b\}$ and the following set of integrity constraints I:

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

$$\leftarrow not\, q, not\, b \tag{4}$$

From the above abductive framework $\langle T, A, I\rangle$, we can get $M_1(\Delta_1) = \{b, p\}$ where $\Delta_1 = \{b\}$ and $M_2(\Delta_2) = \{a, q\}$ where $\Delta_2 = \{a\}$ as generalized stable models.

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

$$a \leftarrow not\, \widetilde{a} \tag{5}$$

$$\widetilde{a} \leftarrow not\, a \tag{6}$$

$$b \leftarrow not\, \widetilde{b} \tag{7}$$

$$\widetilde{b} \leftarrow not\, b \tag{8}$$

We can see that the following two sets of propositions are actually stable models of $\langle T \cup \Gamma(A), I\rangle$:

$M'_1 = M_1(\Delta_1) \cup \widetilde{\nabla_1} = \{b, p\} \cup \{\widetilde{a}\} = \{b, p, \widetilde{a}\}$ and

$M'_2 = M_2(\Delta_2) \cup \widetilde{\nabla_2} = \{a, q\} \cup \{\widetilde{b}\} = \{a, q, \widetilde{b}\}$. □

We have another correspondence with respect to hypotheses used for explanation.

Theorem 2 *Let $\langle T, A, I\rangle$ be an abductive framework and $\langle T \cup \Gamma(A), I\rangle$ be its translation and q be an observation and I' be $I \cup \{\leftarrow not\, q\}$. q has an explanation with a set of hypotheses Δ if and only if there is a stable model*

M' for $\langle T \cup \Gamma(A), I' \rangle$ such that $\Delta = M' \cap A$.

Proof: See Appendix.

Example 2 *Explanation*

Consider the abductive framework in Example 1. Suppose an observation q is given. This observation has the unique explanation with a set of hypotheses $\{a\}$. Let I' be $I \cup \{\leftarrow not\ q\}$ and consider $\langle T \cup \Gamma(A), I' \rangle$. Then, $M_2' = \{a, q, \tilde{b}\}$ is the unique stable model for $\langle T \cup \Gamma(A), I' \rangle$ and we can see that $M_2' \cap A = \{a\}$ is equal to the set of hypotheses. □

3 Computing Stable Models for Logic Program with Integrity Constraints

In this section, we give a nondeterministic bottom-up procedure to compute stable models for a logic program with integrity constraints. To combine the previous translation and the following procedure, we can calculate abduction.

From the definition of stable models, one might think that it is sufficient to use the procedure of [Saccà 90, Fages90] and remove every stable model which does not satisfy some integrity constraints in order to obtain all stable models. However, we can save search space if we can check integrity constraints during the process of constructing stable models. The following procedure performs not only such dynamic checking of integrity constraints but also active use of integrity constraints to derive some facts. Moreover, we can use integrity constraints introduced for a given observation actively to find hypotheses used in explanation for the observation.

At first, in the figure 1 we give a skeleton of the procedure to show how the procedure works. In the next subsections, we will show the procedure in detail. **Figure 1**

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

In the following subsections, we provide details of subprocedures *select_rule* and *propagate*.

3.1 The Subprocedure of Propagation

The subprocedure *propagate* in figure 2 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 constraints which derive that $\neg q$ is true from the integrity constraint $\leftarrow q$. (by cases 2 and 4)

Figure 2

The sets M_i and $\widetilde{M_i}$ are equivalent to M_i and $\widetilde{M_i}$ in the procedure of [Saccà 90, p.215] except that in our procedure, we check integrity constraints dynamically (the conflict checking in while loop and cases of 3 and 5 in the subprocedure of *propagate*) and $\widetilde{M}_i$ might increase by cases of 2 and 4 in the subprocedure of *propagate*.

3.2 The Subprocedure of Rule Selection

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 to be satisfied. For the integrity constraint, $\leftarrow not\, p$, for example, which is introduced to find the hypotheses in abductive explanation in the previous section, we know p must be in the models. 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 directly derives p. If such a rule does not exist, we had better select a rule which has a possibility of deriving p. Therefore, it is important to consider the integrity constraint of the form of $\leftarrow not\, p$ at early stage of the procedure, especially for computing an abductive explanation for an observation.

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

$$p \leftarrow q, r, not\, s \qquad (1)$$

$$q \leftarrow not\, t \qquad (2)$$

$$r \leftarrow not\, u \qquad (3)$$

Given the integrity constraint, $\leftarrow not\, p$, p must be in the models of the above example. Because only rule (1) has p as its head, rule (1) must be used to derive p. In order for p to be in the 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 the models, so rule (2) has a possibility of deriving p. In a similar way, we find that rule (3) also has a possibility of deriving p. In this way, we can find a rule with a possibility of deriving p in a top-down manner from an integrity constraint of the form $\leftarrow$ *not* p.

To check whether there is a selectable rule and to decide which rule should be selected, the procedure calls the subprocedure *select_rule* (figure 3). *select_rule* calls the subprocedure *topdown_check* (figure 4), which performs top-down expectation. We show these subprocedures as follows.

Figure 3
Figure 4

Initially *select_rule* checks whether there is an integrity constraint for which top-down expectation is performed. More precisely, top-down expectation can be also performed for rules whose heads are already in $\widetilde{M}$. Neg expresses a set of the negated propositions in such an integrity constraint. If there is no Neg, that is $Negs = \emptyset$, top-down expectation is not performed and we must select a rule with no clue. Otherwise, we perform top-down expectation.

In *topdown_check*, by M_t and $\widetilde{M_t}$, a rule is selected which is consistent in the rules previously selected during top-down expectation. Moreover M_u is used to exclude cyclic derivations.

Our procedure computes mainly in a bottom-up manner. Top-down expectation is only 'expectation', since the procedure *topdown_check* doesn't construct a model but returns some clue in order to begin computation of the procedure *propagate*.

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

Theorem 3 *Let $\langle T, I\rangle$ be a normal logic program with integrity constraints.*

1. *If the procedure outputs M, then M is a stable model for $\langle T, I\rangle$.*

2. *If T and I are finite, then the procedure outputs all stable models by an exhaustive search.*

Proof: See Appendix.

4 Examples

We compare our procedure with the procedure of [Saccà 90] with integrity constraint check afterwards. The following example shows the difference.

Example 3 *Difference of Two Procedures*

Consider the following program:

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

$$r \leftarrow not\, q \tag{2}$$

$$q \leftarrow not\, r \tag{3}$$

and the integrity constraint:

$$\leftarrow p \tag{4}$$

The procedure of [Saccà 90] produces stable models $\{p, q\}$ and $\{r\}$ for a logic program of (1), (2) and (3). So, this process has a nondeterminism of producing two stable models. Then, we discard $\{p, q\}$ because this model does not satisfy the integrity constraint (4).

On the other hand, the execution of our procedure is as follows:

0. $M_0 = \{r\}, \widetilde{M}_0 = \{p, q\}$,
   because from (4), p must be in $\widetilde{M}_0$ by case 4 in *propagate*,
   and from (1), q must be in $\widetilde{M}_0$ by case 2 in *propagate*,
   and from (2), r must be in M_0 by case 1 in *propagate*.

1. Since *select_rule* returns *nil_rule*, M_0 is returned.

Therefore, in this example, we can calculate a stable model deterministically in our procedure. Note that, in this execution, the integrity constraint (4) is used to derive that p is out of belief and the rule (1) is used to derive that q is out of belief. So, this example shows an active usage of integrity constraints and a top-down propagation of a disbelieved atom in our procedure. □

Now, we can calculate abduction by combining the translation from an abductive framework into a logic program with integrity constraints and the above procedure to compute stable models for the translated logic program with integrity constraints.

Example 4 *Combination of Translation and Bottom-up Procedure*

We calculate abductive explanation for (1)∼(8) in Example 1 and the observation in Example 2. The execution of our procedure for the program as follows:

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

1. $Negs = \{\{q, b\}, \{q\}\}$ in *select_rule* because of integrity constraints (4) and (9). Since Neg can be an arbitrary set in $Negs$, in this case $Neg = \{q\}$ by heuristics[2]. Select rule (2) and (5) in *topdown_check*$(M_0, \widetilde{M}_0, q)$. 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 *select_rule* returns *nil_rule*, M_1 is returned.

So, we get an abductive explanation $M_1 \cap A = \{a\}$ by the correspondence proved in Section 2.

If we did not consider the top-down expectation, then we would have six alternatives for selecting a rule. Among the six alternatives, there is an exactly same rule selection as above. However, four of other alternatives cannot provide the explanation for the observation q because the alternatives do not select rule (5) which is necessary to derive q. The last alternative results in the same model as the above computation but first selects the irrelevant rule to q. This example makes clear that we can reduce the amount of backtracking thanks to top-down expectation of integrity constraints.

5 Related Work

There are many works to show the methods computing stable models. Among the previous works, in [Fages90, Saccà 90] a well-founded bottom-up proce-

[2]Even if the integrity (4) is satisfied only by b, the integrity constraint for the observation, $\leftarrow$ *not* q, must be satisfied. Therefore, we had better search the possibility of deriving q by *topdown_check*.

dure for calculating stable models forlogic programs *without* integrity constraints is provided. Our procedure can be regarded as an extension of the bottom-up procedure provided in [Fages90, Saccà 90]. The point of the extension is that our procedure deals with integrity constraints and utilizes the integrity constraints to reduce search spaces, especially for abduction by top-down expectation.

Eshghi [Eshghi90] has given an algorithm using ATMS and a filtering mechanism to generate stable models from labels in ATMS. However, he considers only logic programs without integrity constraints.

Giordano and Martelli [Giordano 90] have given a translation of a set of TMS justifications with integrity constraints to another set of justifications without integrity constraints to produce all stable models including stable models obtained by dependency-directed backtracking(DDB). Although this work is important in its own right to give a semantics for DDB of Doyle's TMS, this semantics conflicts with the original usage of integrity constraints in deductive databases, that is, checking violated updates. The problem is that even if an update is violated by the current integrity constraints, we might get other consistent states by performing DDB and therefore, we might not be able to detect a violation of the updates.

From the view point of computing abductive logic programming, there are related works to the proposed method. Kakas and Mancarella [Kakas 90a] have extended Eshghi's top-down procedure [Eshghi89] for abduction so that arbitrary abducibles can be used. Although their procedure is limited for a certain class of logic programs in order to guarantee correctness, they show that their procedure is suitable for a truth maintenance mechanism to manipulate consistent explanations for a series of observations which can be regarded as a non-monotonic extension of ATMS.

6 Conclusion

In this paper, we present a method of calculating abduction by translating an abductive framework into a logic program with integrity constraints and computing stable models for the program.

We have proposed a query evaluation method for an abductive framework in [Satoh 92]. The procedure in [Satoh 92] can be regarded as an extension of the procedure of Kakas and Mancarella by adding forward (or bottom-up) evaluation of rules and consistency check for implicit deletion. In the future, we should investigate and compare the computational complexity of our procedures in this paper and [Satoh 92].

Acknowledgments: We thank Katsumi Inoue from ICOT and Vladimir Lifschitz from Stanford University and University of Texas at Austin for helpful discussions. This work was done while authors were in Institute for New Generation Computer Technology (ICOT).

References

[Doyle 79] Doyle, J., A Truth Maintenance System, *Artificial Intelligence*, **12**, pp. 231 – 272 (1979).

[Elkan 90] 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).

[Eshghi90] Eshghi, K., Computing Stable Models by Using the ATMS *Proc. of AAAI'90*, pp. 272 – 277 (1990).

[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).

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

[Giordano 90] Giordano, L., Martelli, A., Generalized Stable Models, Truth Maintenance and Conflict Resolution *Proc. of ICLP'90*, pp. 427 – 441 (1990).

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

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

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

[Satoh 91] Satoh, K., Iwayama, N., Computing Abduction by Using TMS, *Proc. of ICLP'91*, (1991).

[Satoh 92] Satoh, K., Iwayama, N., A Query Evaluation Method for Abductive Logic Programming, *Proc. of JICSLP'92*, (1992).

Appendix

Proof of Theorem 1:

We first prove the following lemma.

Lemma 1 *Let $\langle T, A, I\rangle$ be an abductive framework and $T' = T \cup \Gamma(A)$ and Δ be a subset of A. Let $M(\Delta)$ be a subset of propositions used in $\langle T, A, I\rangle$ such that $M(\Delta) \cap A = \Delta$. Let M' be $M(\Delta) \cup \widetilde{\nabla}$. Then,*

$$min(T'^{M'}) = min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cup \widetilde{\nabla}$$

where $min(\Upsilon)$ means the minimal model of a logic program Υ.

Proof:

$min(T'^{M'})$

$= min((T \cup \Gamma(A))^{M'})$

$= min(T^{M'} \cup \Gamma(A)^{M'})$

Since T contains only symbols in $M(\Delta)$, $T^{M'} = T^{M(\Delta)}$.

And since $\Gamma(A)$ contains only symbols in $\Delta \cup \widetilde{\nabla}$, $\Gamma(A)^{M'} = \Gamma(A)^{\Delta \cup \widetilde{\nabla}}$.

For every abducible $p \in A$ and for every pair of rules in $\Gamma(A)$,

if $p \in \Delta$ then $\{p \leftarrow not\, \tilde{p}, \tilde{p} \leftarrow not\, p\}^{\Delta \cup \widetilde{\nabla}} = \{p \leftarrow\}$

else if $p \notin \Delta$, that is, $\tilde{p} \in \widetilde{\nabla}$ then $\{p \leftarrow not\, \tilde{p}, \tilde{p} \leftarrow not\, p\}^{\Delta \cup \widetilde{\nabla}} = \{\tilde{p} \leftarrow\}$.

Therefore, $\Gamma(A)^{\Delta \cup \widetilde{\nabla}} = \mathcal{F}(\Delta) \cup \mathcal{F}(\widetilde{\nabla})$ where $\mathcal{F}(\widetilde{\nabla}) = \{\tilde{p} \leftarrow | \tilde{p} \in \widetilde{\nabla}\}$.

Thus,

$min(T^{M'} \cup \Gamma(A)^{M'})$

$= min(T^{M(\Delta)} \cup \mathcal{F}(\Delta) \cup \mathcal{F}(\widetilde{\nabla}))$

$= min((T \cup \mathcal{F}(\Delta))^{M(\Delta)} \cup \mathcal{F}(\widetilde{\nabla}))$ since $\mathcal{F}(\Delta) = (\mathcal{F}(\Delta))^{M(\Delta)}$

$= min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cup \widetilde{\nabla}$

since no common symbols in $T \cup \mathcal{F}(\Delta)$ and $\widetilde{\nabla}$. □

Now we prove Theorem 1.

(1) Assume $M(\Delta) = min((T \cup \mathcal{F}(\Delta))^{M(\Delta)})$ and $M(\Delta)$ satisfies all of integrity constraints in I.

$min(T'^{M'})$

$= min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cup \widetilde{\nabla}$ by Lemma 1,

$= M(\Delta) \cup \widetilde{\nabla}$ by the assumption,

$= M'$

This means that M' is a stable model of T'. Since $M(\Delta) \subseteq M'$, M' also satisfies all of integrity constraints in I.

(2) Assume $M' = min(T'^{M'})$ and M' satisfies all of integrity constraints in I. Let Δ be $M' \cap A$ and $M(\Delta)$ be $M' - \widetilde{\nabla}$.

By Lemma 1, $min(T'^{M'}) = min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cup \widetilde{\nabla}$.

And since $M(\Delta) = M' - \widetilde{\nabla}$, $M' = M(\Delta) \cup \widetilde{\nabla}$.

Therefore, by the assumption, $min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cup \widetilde{\nabla} = M(\Delta) \cup \widetilde{\nabla}$.

Since $(min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) \cap \widetilde{\nabla}) = \emptyset$ and $(M(\Delta) \cap \widetilde{\nabla}) = \emptyset$, $min((T \cup \mathcal{F}(\Delta))^{M(\Delta)}) = M(\Delta)$.

This means that $M(\Delta)$ is a stable model of $T \cup \mathcal{F}(\Delta)$. Since every integrity constraint in I uses only propositions in T and A which receive the same interpretation in M' and $M(\Delta)$, $M(\Delta)$ also satisfies all of integrity constraints in I. □

Proof of Theorem 2:

Suppose $M(\Delta)$ is a generalized stable model for $\langle T, A, I\rangle$ and $q \in M(\Delta)$. This means $M(\Delta) \models (\leftarrow \mathit{not}\, q)$. Therefore, $M(\Delta)$ is also a generalized stable model for $\langle T, A, I'\rangle$. From Theorem 1, there exists a stable model M' for $\langle T \cup \Gamma(A), I'\rangle$ such that $M' = M(\Delta) \cup \widetilde{\nabla}$. Thus, $M' \cap A = \Delta$.

Suppose M' is a stable model for $\langle T \cup \Gamma(A), I'\rangle$. Let Δ be $M' \cap A$ and $M(\Delta)$ be $M' - \widetilde{\nabla}$. From Theorem 1, $M(\Delta)$ is a generalized stable model for $\langle T, A, I'\rangle$ and therefore, $q \in M(\Delta)$ □

Proof of Theorem 3:

Consider the simple procedure in figure 5 to compute a stable model. **Figure 5**

We denote our procedure in Section 3 as *proc*(O) and denote the above simple procedure as *proc*(S). We show that outputs of *proc*(O) and *proc*(S) are equal (Lemma 2) and outputs of *proc*(S) are actually all stable models of a given logic program with integrity constraints (Lemma 4).

Firstly, we show the following lemma. We say that *proc*(S) *outputs* M *with a sequence of rules* $R_0, ..., R_n$ if $R_0, ..., R_n$ are selected in this order in Step 1 of *proc*(S), and *proc*(S) outputs M after selecting these rules. Similarly, we say that *proc*(O) *outputs* M *with a sequence of rules* $R_0, ..., R_n$ if $R_0, ..., R_n$ are selected in this order in Step 1 in the main procedure of *proc*(O) and case 1 in *propagate*, and *proc*(O) outputs M after selecting these rules.

Lemma 2

1. *If proc(O) outputs M with a sequence of rules then proc(S) outputs M with the same sequence of rules.*

2. Let T and I be finite. If proc(S) outputs M with a sequence of rules then there exists a sequence of rules with which proc(O) outputs M.

Proof of Lemma 2:

1. We can show that the sequence of rules by *proc*(O) can be selected along iterations of *proc*(S) to output M.

2. Let $\langle T, I\rangle$ be a finite normal logic program with finite integrity constraints. Suppose *proc*(S) outputs M with a sequence $R_0, ..., R_n$. We show by induction on the numbers of i of iterations of the main procedure in *proc*(O) and k of iterations of *propagate* in *proc*(O) that the following conditions hold:

Condition 1: There are two sequences $s_{modified}$ and s_{rest} such that

> 1-(a) *proc*(S) outputs M with $s_{modified} \cdot s_{rest}$ where $\cdot$ is a concatenation of two sequences, and
>
> 1-(b) we can select rules along $s_{modified}$ in *proc*(O) up to i and k.

Condition 2: $(\widetilde{M}_i^k \cap M) = \emptyset$

Note that a set of propositions constructed in *proc*(S) up to $s_{modified}$ is equal to a set constructed in *proc*(O) up to $s_{modified}$ by Condition 1 and so, the set constructed in *proc*(O) is a subset of M.

If $i = 0$ and $k = 0$, let $s_{modified} = \langle\rangle$ (null sequence) and $s_{rest} = \langle R_0, ..., R_n\rangle$ (the original sequence).
Then, Condition 1 clearly holds. And since $\widetilde{M}_i^k = \emptyset$, Condition 2 holds.

Suppose up to i and k, the above conditions hold. We enter an iteration of *propagate*. We show the above conditions hold for i and $k + 1$.

Condition 1-(a): This condition should be checked if there is a rule which satisfies case 1 of *propagate*. Suppose there is such a rule R, that is, $pos(R) \subseteq M_i^k$ and $neg(R) \subseteq \widetilde{M}_i^k$. Then, $pos(R) \subseteq M$ and $(neg(R) \cap M) = \emptyset$ since M_i^k is equal to a set of propositions constructed by $s_{modified}$ in $proc(S)$ and $(\widetilde{M}_i^k \cap M) = \emptyset$ by the inductive hypothesis. Therefore, R can be selected after this point by $proc(S)$ until $head(R)$ is added. Since $head(R)$ is included in M, there must be a rule R' in s_{rest} such that $head(R') = head(R)$. We delete R' from s_{rest} and add R to the tail of $s_{modified}$ to obtain new s_{rest} and new $s_{modified}$ for i and $k+1$.

Now, it is sufficient to show that $proc(S)$ outputs M with $s_{modified} \cdot s_{rest}$. Suppose $proc(S)$ does not output M with $s_{modified} \cdot s_{rest}$. There are two possibilities for this situation.

1. There is some R'' in s_{rest} which contains $head(R)$ in $neg(R'')$.
2. There is some R'' in s_{rest} whose head is contained in $neg(R)$.

The first case is impossible since if such R'' is equal to R' or appears before R' in the previous s_{rest}, R' can not be selected in the previous s_{rest}, and if such R'' appears after R' in the previous s_{rest}, R'' can not be selected in the previous s_{rest}. The second case is also impossible since $(\widetilde{M}_i^k \cap M) = \emptyset$ by the inductive hypothesis and $head(R'') \in M$ and $neg(R) \subseteq \widetilde{M}_i^k$.

Thus, Condition 1-(a) holds for i and $k+1$.

Condition 1-(b): As shown above, if there is a rule which satisfies case 1 of *propagate*, we can choose the rule along $s_{modified}$ by $proc(O)$.

However, we should also check if there is a rule satisfying case 3 or there is an integrity constraint satisfying case 5 since if there is such a rule or such an integrity constraint, $proc(O)$ will fail.

Suppose there is a rule R satisfying case 3. Then, R must be eventually selected by $proc(S)$ since $(\widetilde{M}_i^k \cap M) = \emptyset$ by the inductive hypothesis or $head(R)$ will be added by some other rule. In either case, $head(R)$ must be included in M and this leads to contradiction because $head(R) \in \widetilde{M}_i^k$, and $(\widetilde{M}_i^k \cap M) = \emptyset$. Therefore, there is no rule satisfying case 3.

Suppose there is an integrity constraint C satisfying case 5. Then, since $(\widetilde{M}_i^k \cap M) = \emptyset$, M does not satisfy C and so, $proc(S)$ will fail at the end of the execution. Therefore, there is no integrity constraint satisfying case 5.

Thus, Condition 1-(b) holds for i and $k+1$.

Condition 2: This condition should be checked if there is a proposition added to $\widetilde{M}_i^{k+1}$ by case 2 or case 4.

Suppose that there is a proposition P added to $\widetilde{M}_i^{k+1}$ by case 2, that is, $P \notin M_i^k$ and there is a rule R such that $head(R) \in \widetilde{M}_i^k$ and $P \in pos(R)$ and $(pos(R) - \{P\}) \subseteq M_i^k$ and $neg(R) \subseteq \widetilde{M}_i^k$. Then, we show that M does not include P. Suppose M includes P. R must be eventually selected by $proc(S)$ since $(\widetilde{M}_i^k \cap M) = \emptyset$ by the inductive hypothesis or $head(R)$ will be included by some other rule. In either case, $head(R)$ must be included in M and this leads to contradiction because $head(R) \in \widetilde{M}_i^k$, and $(\widetilde{M}_i^k \cap M) = \emptyset$. Therefore, M does not include P.

Suppose that there is a proposition P added to $\widetilde{M}_i^{k+1}$ by case 4, that is, $P \notin M_i^k$ and there is an integrity constraint C such that $P \in pos(C)$ and $(pos(C) - \{P\}) \subseteq M_i^k$ and $neg(C) \subseteq \widetilde{M}_i^k$. Then, we show that M does not include P. Suppose M includes P. Then, since $(\widetilde{M}_i^k \cap M) = \emptyset$, M does not satisfy C and so, $proc(S)$ will fail at the end of the execution. Therefore, M does not include P.

Thus, Condition 2 holds for i and $k + 1$.

If this iteration (i and $k + 1$) is not the last iteration in *propagate*, we can also prove that the above conditions hold for a new iteration (i and $k + 2$). Otherwise, we return to the while loop and go to the if sentence. We cannot fail at the if sentence in the while loop since $M_i^{k+1} \subseteq M$ and $(\widetilde{M}_i^{k+1} \cap M) = \emptyset$. Therefore, we can go to *select_rule* in the while loop.

In the case that $s_{rest} = \langle\rangle$, we assume there exists a non empty set which satisfies one of the two conditions of Neg in *select_rule*. This contradicts that $proc(S)$ outputs an answer with $s_{modified} \cdot s_{rest}$. So $Negs = \emptyset$ if $s_{rest} = \langle\rangle$. Since *select_rule* returns *nil_rule* in this case, $proc(O)$ outputs M with $s_{modified}$.

We show that *select_rule* returns some rule, if $s_{rest} \neq \langle\rangle$. If $Negs = \emptyset$ at *select_rule*, the then part of *select_rule* can return the left-most rule in s_{rest}. In this case, we delete the left-most rule in s_{rest} and add the rule to the tail of $s_{modified}$.

Next we consider the other case that $Negs \neq \emptyset$ at *select_rule*. At first we show that *topdown_check* returns some rule R from s_{rest}, which is not necessarily the left-most rule in s_{rest}. Since, in the next lemma, we see that the *topdown_check* returns a rule in s_{rest}, we can confirm that Conditions 1 and 2 hold for $i+1$ and 0 after we delete the returned rule R by *topdown_check* and add R to the tail of $s_{modified}$.

Lemma 3 *topdown_check can return a rule in* s_{rest}.

Proof: We show the following propositions which are enough to show the lemma.

1. *topdown_check* doesn't fail and can select R_j from s_{rest} at select.
2. If R_j and R_{j+1} are in s_{rest}, $th(R_j) > th(R_{j+1})$. ($th(R) = n$ means R is the n-th rule in s_{rest}, and j is the number of the iteration of the **goto** loop in *topdown_check*.)

1. We show by induction on j. If $j = 0$, that means $Pos = \{p\}$, there must be a rule in s_{rest} whose head is p (the case where there isn't such a rule contradicts the fact that $proc(S)$ outputs the answer with $s_{modified} \cdot s_{rest}$).

Suppose up to j, *topdown_check* doesn't fail. If $pos(R_j) - M \neq \emptyset$, every proposition $pr \in pos(R_j) - M$ should have a rule R in s_{rest} which satisfies $head(R) = pr$ and the other three conditions at the **if** sentence (otherwise R_j cannot be selected in s_{rest} by $proc(S)$). So we can select the rule from s_{rest} at select in *topdown_check*.

2. By 1 all R_j can be in s_{rest}. Suppose $th(R_j) = th(R_{j+1})$, which means $R_j = R_{j+1}$. Then, for some $p \in pos(R_j) - M$, $head(R_{j+1}) = p$. This contradicts $head(R_j) \notin pos(R_j)$.

Suppose $th(R_j) < th(R_{j+1})$. There exists $p \in pos(R_j) - M$ s.t. $head(R_{j+1}) = p$. But this contradicts that $proc(S)$ outputs the answer with $s_{modified} \cdot s_{rest}$.

Finally, if every R_j is selected from s_{rest}, *topdown_check* eventually terminates, because $th(R_j) > 0$ for all j and $th(R_j) > th(R_{j+1})$. So the lemma is proved. □

Finally, we show the following lemma.

Lemma 4

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

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

To show Lemma 4, we need the following definition of a *finite grounded model.*

Definition 7 *Let $\langle T, I\rangle$ be a logic program with integrity constraints. A set of propositions M is a* finite grounded model *for $\langle T, I\rangle$ if the following are satisfied.*

1. *M is a model of T.*

2. *M satisfies every $C \in I$.*

3. *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 $pos(R_j) \subseteq \{P_1, ..., P_{j-1}\}$ 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 every propositions in M, $\langle R_1, R_2, ...R_n \rangle$, is* a sequence of supporting rules *for M.*

We can prove the following lemma by extending [Elkan 90, Theorem 3.8].

Lemma 5 *Let $\langle T, I \rangle$ be a logic program with integrity constraints. A set of propositions M is a finite grounded model for $\langle T, I \rangle$ if and only if M is a finite stable model for $\langle T, I \rangle$.*

Proof of Lemma 4:

1. If $proc(S)$ outputs M with a finite sequence of selected rules $R_0, ..., R_n$, then this sequence actually gives a sequence of supporting rules. We can show that M is a model of T and M satisfies every $C \in I$. Therefore, M is a finite grounded model and, so, a finite stable model for $\langle T, I \rangle$ by Lemma 5.

2. Let $\langle T, I \rangle$ be a finite logic program with finite integrity constraints. Suppose M is a stable model for $\langle T, I \rangle$. Since $\langle T, I \rangle$ is a finite logic program with finite integrity constraints, M is a finite stable model and therefore, a finite grounded model by Lemma 5. Then, there is a finite sequence of supporting rules $R_0, ..., R_n$. We can show that $proc(S)$ outputs M with this sequence of rules. Since all sequences of rules can be selected by exhaustive search, $proc(S)$ outputs all stable models. □

Theorem 3 is proved by Lemma 2 and Lemma 4. □

図表

Let $\langle T, I \rangle$ be a normal logic program with integrity constraints.

```
$i := 0$,
$M_0, \widetilde{M_0} := propagate(\emptyset, \emptyset)$.
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 $M_i$.
```

Figure 1: A Procedure to Compute a Stable Model (skeleton)

```
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 $R$ in $T$

    1. If $head(R) \notin M_i^{k-1}$ 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 $head(R)$ to $M_i^k$.

    2. If $head(R) \in \widetilde{M}_i^{k-1}$ 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 $head(R) \in \widetilde{M}_i^{k-1}$ 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 $C$ 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
```

Figure 2: The Subprocedure of *propagate*

```
procedure select_rule
$Negs := \{Neg | Neg$ is a non empty set of propositions satisfying one of the following conditions$\}$

    1. there exists a rule $R$ in $T$ satisfying the following conditions
        (a) $Neg = \{n | n \in neg(R)$ and $n \notin M_i\}$,
        (b) $head(R) \in \widetilde{M_i}$,
        (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) $Neg = \{n | n \in neg(C)$ and $n \notin M_i\}$,
        (b) For every $p \in pos(C), p \in M_i$,
        (c) For every $n \in neg(C), n \notin M_i$.

If $Negs = \emptyset$ then
    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
else
    $Neg$ := an arbitrary set in $Negs$,
    select a proposition $pr$ from $Neg$,
    return topdown_check($M_i, \widetilde{M_i}, pr$)
endif
```

Figure 3: The Subprocedure of *select_rule*

```
procedure $topdown\_check(M, \widetilde{M}, p)$
$j := 0$,
$M_t := \widetilde{M_t} := M_u := \emptyset$,
$Pos := \{p\}$,

Label:
if for every proposition $pr \in Pos$, there is a rule $R$ in $T$ which satisfies the following
conditions

    1. $head(R) = pr$,

    2. $head(R) \notin pos(R)$, $head(R) \notin neg(R)$,

    3. For every $p \in pos(R), p \notin \widetilde{M}$, $p \notin \widetilde{M_t}$ and $p \notin M_u$,

    4. For every $n \in neg(R), n \notin M$ and $n \notin M_t$.

then

    $pr_j$ := a arbitrary proposition in $Pos$
    $Rset := \{R | R$ in $T$ satisfies the above conditions and $head(R) = pr_j\}$
    select a rule $R_j$ from $Rset$,
    $Pos := pos(R_j) - M$,
    if $Pos = \emptyset$ then return $R_j$
    else

        $M_t := M_t \cup pos(R_j)$,
        $\widetilde{M_t} := \widetilde{M_t} \cup neg(R_j)$,
        $M_u := M_u \cup head(R_j)$,
        $j := j + 1$,
        goto Label

else fail
```

We express $\{p | p \in pos(R_j)$ and $p \notin M\}$ as $pos(R_j) - M$.

Figure 4: The Subprocedure of *topdown_check*

```
Let $\langle T, I \rangle$ be a normal logic program with integrity constraints.
$i := 0$

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.
```

Figure 5: A Simple Procedure to Compute a Stable Model