ICOT Technical Report: TR-722

TR-722

Embedding Negation as Failure into a Model Generation Theorem Prover

by
K. Inoue, M. Koshimura & R. Hasegawa

December, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Embedding Negation as Failure into a Model Generation Theorem Prover

Katsumi Inoue Miyuki Koshimura* Ryuzo Hasegawa
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
phone: +81-3-3456-2514
email: inoue@icot.or.jp, koshi@icot.or.jp,
hasegawa@icot.or.jp
November 11, 1991

Abstract

Here, for the first time, we give an implementation which computes answer sets of every class of logic program and deductive database. The proposal is based on bottom-up, incremental, backtrack-free computation of the minimal models of positive disjunctive programs, together with integrity constraints over beliefs and disbeliefs. The procedure has been implemented on top of a model generation theorem prover (MGTP) on a distributed-memory multiprocessor machine (Multi-PSI), and is currently being applied to a legal reasoning system developed at ICOT.

1 Introduction

This paper presents a novel and simple procedure which computes the models of logic programs containing *negation-as-failure* formulas. In traditional top-down proof procedures such as SLDNF-resolution, *not* P succeeds if there is no top-down proof of P; the meaning of negation as failure is only procedural. On the other hand, in recent theories of logic programming and deductive databases, declarative semantics have been given to extensions of logic programs, where the negation-as-failure operator is considered to be a *nonmonotonic* modal operator [7, 8, 14]. In particular, logic programs or deductive databases containing both negation as failure and classical negation can be interpreted as Reiter's default theories [18] or disjunctive default theories [10]. With these new semantics, logic programming can be used as a powerful knowledge representation tool, whose applications contain reasoning with incomplete knowledge [8, 9], expression of "don't-care" nondeterminism [19], exception handling [13], default reasoning and abduction [11].

However, for these extended classes of logic programs, the top-down ap-

*Presently at: Toshiba Information Systems

proach cannot be used for their computation because there is no local property in evaluating programs. For example, there has been *no* top-down proof procedure which is sound with respect to *stable model semantics* [7] for general logic programs. Thus we need *bottom-up* computation for evaluation of the nonmonotonic operator *not*. This area is progressing, and there have been some proposals for computing stable models of general logic programs [19, 3, 21]. Unfortunately, these previous approaches are only applicable to a simple class of programs so that it is difficult to deal with disjunctive databases.

We show a bottom-up computation of *answer sets* for any class of logic programs, including the *extended disjunctive databases* proposed by Gelfond and Lifschitz [9] the proof procedure of which has not been found. Bottom-up computation reasons forwards, starting from unconditional literals, accumulating proved literals, and outputting models of programs. In evaluating *not* P in a bottom-up manner, it is necessary to interpret *not* P with respect to a fixpoint of computation because even if P is not currently proved, P might be proved in subsequent inferences. We thus come up with a completely different way of thinking for *not*: when we have to evaluate *not* P in a current world, or a partial model, instead of computing "negation by failure to prove P", we split the world into two: (1) the world where P is assumed not to hold, and (2) the world where it is necessary that P holds. Each negation-as-failure formula *not* P is thus translated into negative and positive literals with a modality expressing belief, i.e., "disbelieve P" and "believe P".

The basic idea of bottom-up computation is thus to translate any logic program (with negation as failure) to a *positive disjunctive program* (without negation as failure) [16] of which a *model generation theorem prover*, like SATCHMO [15] or the MGTP [6], can compute the minimal models. Some pruning rules with respect to "believed" or "disbelieved" literals are expressed as integrity constraints that are dealt with by using object-level schemata on the MGTP [12]. The MGTP then finds *all* answer sets *incrementally*, *without backtracking*, and *in parallel*. The proposed method is surprisingly simple and does not increase the computational complexity of the problem more than computation of the minimal models of positive disjunctive programs. The procedure has been implemented on top of the MGTP on a distributed-memory multiprocessor machine (Multi-PSI), and is currently being applied to a legal reasoning system. While we use the MGTP to generate models in this paper, the proposed transformation method can be linked with any method to compute models [5, 1] or a fixpoint construction like [17].

2 Positive Disjunctive Programs

This section shows how the MGTP [6] computes the minimal models of a *positive disjunctive program*, that is, a disjunctive database [9] which contains neither negation as failure nor classical negation. The MGTP can deal with this class of programs, and in later sections every other extended program can be shown to be translated to a positive disjunctive program.

2.1 Minimal Models

A *positive disjunctive program* [16] is a set of rules of the form:

$$A_1 \mid \ldots \mid A_l \leftarrow A_{l+1}, \ldots, A_m\,, \tag{1}$$

where $m \geq l \geq 0$ and each A_i is an atom. According to [9], we use the connective "|" instead of "$\vee$" although each A_i $(l \geq i \geq 1)$ is a disjunct of the consequent of the rule. When $l = 0$, a rule of the form

$$\leftarrow A_1, \ldots, A_m \tag{2}$$

is called an *integrity constraint* [1].

The meaning of a positive disjunctive program Σ can be given by the *minimal models* of Σ [16]. We represent the semantics in a similar way to the definition given by Gelfond & Lifschitz [9], as follows. A rule containing variables stands for the set of its ground instances. We denote the set of ground literals in the language with $\mathcal{L}$. An *answer set* of Σ is any minimal subset S of $\mathcal{L}$ such that:

1. For any ground rule $A_1 \mid \ldots \mid A_l \leftarrow A_{l+1}, \ldots, A_m$ $(l \geq 1)$ of Σ,
   if $A_{l+1}, \ldots, A_m \in S$, then for some i $(1 \leq i \leq l)$, $A_i \in S$;

2. For any ground integrity constraint $\leftarrow A_1, \ldots, A_m$ of Σ,
   if $A_1, \ldots, A_m \in S$, then $S = \mathcal{L}$.

We say Σ is *contradictory* if it has the answer set $\mathcal{L}$. It is easy to see that a contradictory program has the unique answer set $\mathcal{L}$. Unless a program Σ is contradictory, any answer set of Σ is a set of ground atoms, and the set of answer sets of Σ is equivalent to the set of minimal models of the program when each rule of the form (1) in Σ is identified with a clause:

$$A_1 \vee \ldots \vee A_l \vee \neg A_{l+1} \vee \ldots \vee \neg A_m\,.$$

Also, Σ is contradictory if and only if Σ is unsatisfiable in its clausal form.

[1] We allow for integrity constraints, that is, rules with empty consequents, in every class of logic programs and deductive databases. While this form of rules is not excluded by Gelfond & Lifschitz's definitions [7, 8, 9], the corresponding semantics are not explicitly described.

2.2 MGTP

The answer sets or the minimal models of positive disjunctive programs can be computed by using the MGTP [6]. The MGTP is a parallel and refined version of SATCHMO [15], which is a bottom-up model generation theorem prover that uses hyperresolution and case-splitting on non-unit derived clauses. In order to emphasize that the MGTP computes the models in a bottom-up manner, we express each rule of the form (1) in a positive disjunctive program as

$$A_{l+1}, \ldots, A_m \rightarrow A_1 \mid \ldots \mid A_l \,. \tag{3}$$

Given a positive disjunctive program Σ, the MGTP extends *model candidates* $\mathcal{S}$ as follows. A model candidate is a subset of $\mathcal{L}$ and the initial set $\mathcal{S}_0$ of model candidates is given as $\{\emptyset\}$. Let $\mathcal{S}$ be a set of model candidates in some stage. The MGTP applies the following two operations to $\mathcal{S}$.

1. For any $S \in \mathcal{S}$ and any ground rule in Σ of the form:

$$A_{l+1}, \ldots, A_m \rightarrow A_1 \mid \ldots \mid A_l \;\; (l \geq 1),$$

   if $A_{l+1}, \ldots, A_m \in S$ and $A_1, \ldots, A_l \notin S$, then remove S from $\mathcal{S}$, and add $S \cup \{A_i\}$ to $\mathcal{S}$ for every $i = 1, \ldots, l$;

2. For any $S \in \mathcal{S}$ and any ground integrity constraint in Σ of the form:

$$A_1, \ldots, A_m \rightarrow \;,$$

   if $A_1, \ldots, A_m \in S$, then remove S from $\mathcal{S}$.

The above two operations correspond to two cases in the definition of answer sets. If the MGTP cannot apply any operation, it stops and returns the model candidates $\mathcal{S}$ in the last stage.

In the above two operations of the MGTP, we can deal with variables more elegantly. Instead of using ground instances of the rules, for each rule with variables in the form (3) we can obtain a substitution σ such that $A_{l+1}\sigma, \ldots, A_m\sigma$ $(l \geq 0)$ is satisfied in a model candidate S. We call the process of obtaining such a substitution σ a *conjunctive matching* of the antecedent literals against elements in S. Note that this process does not need full unification if the *range-restrictedness* condition [15] is imposed on the rules. A rule is said to be range-restricted if every variable in the rule has at least one occurrence in its antecedents. It is sufficient to consider one-way unification, i.e., matching, instead of full unification with occurs check since every model candidate constructed by the extension operation should contain

only ground atoms. This is also a nice property for the implementation of the MGTP in KL1 (the kernel language for parallel inference machine developed in ICOT), because KL1 head unification is simply matching. The MGTP also improves the efficiency by removing redundant conjunctive matching with a *ramified-stack* algorithm [6].

When there are more than two rules whose antecedents are exactly the same, we do not want to perform the same conjunctive matching more than once. For this purpose, the MGTP allows rules of the following form:

$$A_{l+1}, \ldots, A_m \rightarrow A_{1,1}, \ldots, A_{1,k_1} \mid \ldots \mid A_{l,1}, \ldots, A_{l,k_l}, \tag{4}$$

where $m \geq l \geq 0$, $k_i \geq 1$ $(1 \leq i \leq l)$, and each A_i or $A_{i,j}$ is an atom. Each $A_{i,1}, \ldots, A_{i,k_i}$ represents a conjunction of atoms. We call a rule of this form (4) an *MGTP rule*. In summary, the two MGTP operations are formally defined as follows. Let $\mathcal{S}$ be a set of model candidates in some stage, and S any element of $\mathcal{S}$.

1. **(Model candidate extension)** If there is an MGTP rule of the form

   $$A_{l+1}, \ldots, A_m \rightarrow A_{1,1}, \ldots, A_{1,k_1} \mid \ldots \mid A_{l,1}, \ldots, A_{l,k_l} \quad (l \geq 1)$$

   and a substitution σ such that $A_{l+1}\sigma, \ldots, A_m\sigma \in S$, and it does not hold that $A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma \in S$ for any $i = 1, \ldots, l$, then remove S from $\mathcal{S}$, and add $S \cup \{A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma\}$ to $\mathcal{S}$ for all $i = 1, \ldots, l$;

2. **(Model candidate rejection)** If there is an MGTP rule of the form

   $$A_1, \ldots, A_m \rightarrow$$

   and a substitution σ such that $A_1\sigma, \ldots, A_m\sigma \in S$, then remove S from $\mathcal{S}$.

The *MGTP* is defined as a fixpoint operator, whose inputs are a set Σ of MGTP rules and an initial set $\mathcal{S}_0$ of model candidates (usually $\mathcal{S}_0 = \{\emptyset\}$), and whose output $MGTP(\Sigma, \mathcal{S}_0)$ is the set of model candidates closed under the above two operations.

In the following, we assume that function symbols in the language are only constants and that the number of constants is finite. When a set Σ of MGTP rules satisfies these assumptions as well as the range-restrictedness, we say Σ is *finitely groundable* [2]. It is easy to see that a finitely groundable program is decidable. Let us denote the set of minimal (in the sense of set inclusion of literals) model candidates by $min(\mathcal{S})$. For any finitely groundable set Σ of MGTP rules, the following properties can be shown to hold.

Proposition 2.1 If Σ is not contradictory, $min(MGTP(\Sigma, \{\emptyset\}))$ is equivalent to the answer sets of Σ. □

Corollary 2.2 $MGTP(\Sigma, \{\emptyset\}) = \emptyset$ if and only if Σ is contradictory. □

It is guaranteed that a finitely groundable set of rules has at least one minimal model if it is satisfiable [2]. The next is the basic property of the MGTP as a theorem prover.

Corollary 2.3 Suppose that Σ can be identified with the corresponding set of clauses. $MGTP(\Sigma, \{\emptyset\}) = \emptyset$ if and only if Σ is unsatisfiable. □

3 General Logic Programs

This section presents how to compute the answer sets of a *general logic program* [7], that is, a logic program which contains negation-as-failure formulas but does not contain classical negation. While this class of logic programs is an instance of the more general class of disjunctive databases [9] introduced in the next section, we shall first explain the basic idea of bottom-up computation of negation as failure in this section.

A *general logic program* is a set of rules of the form:

$$A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{5}$$

where, $n \geq m \geq l \geq 0$, $1 \geq l \geq 0$, and each A_i is an atom. The meaning of a general logic program is given by the *stable model semantics* [7]. We present the semantics in a similar way to the definition of [10], instead of using the program reduction given in [7]. Let Π be any general logic program, and S a subset of $\mathcal{L}$. S is an *answer set* (or *stable model*) of Π if it coincides with the smallest subset S' of $\mathcal{L}$ such that:

1. For any ground instance of any rule of Π

   $$A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \ \ (l = 1),$$

   if $A_{l+1}, \ldots, A_m \in S'$ and $A_{m+1}, \ldots, A_n \notin S$, then $A_l \in S'$;

2. For any ground instance of any integrity constraint of Π

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

   if $A_1, \ldots, A_m \in S'$ and $A_{m+1}, \ldots, A_n \notin S$, then $S' = \mathcal{L}$.

Again, unless Π is contradictory, that is, if Π does not have the answer set $\mathcal{L}$, every stable model of Π is a set of ground atoms. Unlike positive disjunctive programs, a general logic program may not have any answer set. We say Π is *incoherent* if it has no answer set. Thus, any program is either a *consistent* program (which has consistent answer sets), contradictory program, or incoherent program [11].

Notice that the above definition of stable models is not constructive; S is defined by using itself so that a negation-as-failure formula *not* P is true if P is not true in S. This S can be considered as a *guess* of a possible answer set. If S coincides with the smallest set of atoms deductively closed under the rules of Π, then the guess is correct so that it is acceptable as an answer set. Hence, the most direct way to compute the stable models of Π is to generate all possible guesses and then test if each guess is correct. This method is too explosive to realize because we have to generate and test $2^{|\mathcal{A}|}$ sets of atoms, where $\mathcal{A}$ is the set of ground atoms.

We thus make the number of guesses as few as possible. Let Π_P be the set of rules in Π that do not contain *not*, and Π_N the rest of the rules in Π. If Π has consistent stable models, then every stable model S should contain the least model M of Π_P. To compute the rest of the atoms in S, we make guesses as to whether each atom P appearing as *not* P in Π_N is present. These guesses are delayed as long as possible: if there is a rule in which all antecedents that do not contain *not* are satisfied by a set S' of atoms such that $S' \supseteq M$, then we make a guess with respect to every *not* P in the antecedents, and extend S' by its consequent together with the guess that all such P's will not be present.

Now, we are ready to compute the stable models of Π by using the MGTP. Based on the above discussion, we translate each rule in Π of the form:

$$A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

to the following MGTP rule:

$$A_{l+1}, \ldots, A_m \rightarrow \neg \mathsf{K} A_{m+1}, \ldots, \neg \mathsf{K} A_n, A_l \,|\, \mathsf{K} A_{m+1} \,|\, \ldots \,|\, \mathsf{K} A_n \,. \qquad (6)$$

The intuitive meaning of this MGTP rule is as follows. $\mathsf{K}A$ is a guess that A should hold, or A is *believed*, and $\neg\mathsf{K}A$ is a guess that A should not hold, or A is *disbelieved*. In other words, $\mathsf{K}A$ ($\neg\mathsf{K}A$) imposes the condition that A must hold (A must not hold). For any MGTP rule of the form (6), if a model candidate S' satisfies $A_{l+1}, \ldots, A_m$, then S' is split into $n - m + l$ ($n \geq m \geq 0,\ 0 \leq l \leq 1$) model candidates. In the case of $l = 1$, one of the

split model candidates is assumed that any of $A_{m+1}, \ldots, A_n$ is not satisfied so that the consequent A_l is satisfied. Each of the rest of $n-m$ model candidates is assumed that one of A_i $(m+1 \leq i \leq n)$ is satisfied.

We might relate the symbol K introduced in literals $\mathsf{K}A$, $\neg\mathsf{K}A$ with the modal operator K in an epistemic logic. We call these literals K-*literals*, and other literals without the symbol K *objective literals*. However, we avoid defining the symbol K either as a new connective or as a modal operator since we would like to remain within the MGTP calculus, or positive disjunctive programs, so that both positive and negative K-literals can be dealt with as atoms. To do so, we need to give the conditions which K-literals should satisfy. The following two schemata are thus introduced to reject model candidates when their guesses turn out to be wrong:

- If some ground instance of some atom A holds in S' and is not believed in S', then reject S'.

$$\neg\mathsf{K}A,\ A \rightarrow \quad \text{for every atom } A\,. \tag{7}$$

- If some ground instance of some atom A is believed in S' and is not believed in S', then reject S'.

$$\neg\mathsf{K}A,\ \mathsf{K}A \rightarrow \quad \text{for every atom } A\,. \tag{8}$$

Given a general logic program Π, we denote by $tr_1(\Pi)$ the set of rules consisting of the two schemata (7) and (8), and the MGTP rules obtained by replacing each rule (5) of Π with a rule (6).

The MGTP then computes the fixpoint $MGTP(tr_1(\Pi), \{\emptyset\})$ of model candidates. Each model candidate output by the MGTP contains K-literals as well as objective literals. Next is the condition that all of guesses made so far in a model candidate are correct. Let be $S' \in MGTP(tr_1(\Pi), \{\emptyset\})$.

- If any ground instance A of any atom is believed in S', then it must be true in S':

$$\text{For every ground atom } A, \text{ if } \mathsf{K}A \in S', \text{ then } A \in S'\,. \tag{9}$$

We call condition (9) the **T**-*condition*. It is named after axiom **T** in modal logic. That is, in the fixpoint each K-literal satisfies the condition if the model candidate corresponds to a stable model. Note that the **T**-condition is used as a test and cannot be a schema. We cannot write the condition as

$$\mathsf{K}A \rightarrow A \quad \text{for every atom } A$$

because our K-literals are merely guesses and we should confirm that A is actually derived from the program.

Now, for each model candidate S' computed by the MGTP, we denote the set of literals obtained from S' by removing all K-literals by *objective*(S'). The following four theorems guarantee that the above computation by the MGTP is sound and complete with respect to the stable model semantics.

Theorem 3.1 If a model candidate S' in $MGTP(tr_1(\Pi), \{\emptyset\})$ satisfies the T-condition (9), then $S = objective(S')$ is a stable model of Π. □

Theorem 3.2 If $S \neq \mathcal{L}$ is a stable model of Π, then there is a model candidate S' in $MGTP(tr_1(\Pi), \{\emptyset\})$ such that $S = objective(S')$ and S' satisfies the T-condition (9). □

Theorem 3.3 $MGTP(tr_1(\Pi), \{\emptyset\}) = \emptyset$ if and only if Π is contradictory. □

Theorem 3.4 Π is incoherent if and only if $MGTP(tr_1(\Pi), \{\emptyset\}) \neq \emptyset$ and there is no model candidate which satisfies the T-condition (9). □

In order that each MGTP rule of the form (6) may be *range-restricted*, in the original rule of the form (5) from which the MGTP rule is translated, every variable has at least one occurrence in its antecedents that does not contain *not*. This restriction is as natural as the range-restrictedness in positive disjunctive programs, and can be satisfied in most AI applications. At least, rules can be easily converted in order to satisfy this kind of range-restrictedness.

Example 3.5 Let the general logic program Π_1 consist of the following two rules:

$$P \leftarrow not\, Q\,,$$
$$Q \leftarrow not\, R\,.$$

These two rules are translated to the following MGTP rules:

$$\rightarrow \neg \mathsf{K}Q, P \mid \mathsf{K}Q\,,$$
$$\rightarrow \neg \mathsf{K}R, Q \mid \mathsf{K}R\,.$$

Then, $tr_1(\Pi_1)$ consists of these two rules and the two schemata (7) and (8). Now, let us see how the MGTP computes the stable models of Π_1. We start from the initial model candidates $\mathcal{S}_0 = \{\emptyset\}$.

1. $\mathcal{S}_1 = \{\{\neg \mathsf{K}Q, P\}, \{\mathsf{K}Q\}\}$ by extending $\mathcal{S}_0$ with the first MGTP rule.
2. $\mathcal{S}_2 = \{S_1, S_2, S_3, S_4\}$, where $S_1 = \{\neg \mathsf{K}Q, P, \neg \mathsf{K}R, Q\}$, $S_2 = \{\neg \mathsf{K}Q, P, \mathsf{K}R\}$, $S_3 = \{\mathsf{K}Q, \neg \mathsf{K}R, Q\}$, and $S_4 = \{\mathsf{K}Q, \mathsf{K}R\}$, by extending $\mathcal{S}_1$ with the second MGTP rule.

3. $\mathcal{S}_3 = \{ S_2, S_3, S_4 \}$ by rejecting S_1 with the schema (7).

4. No operation is applicable to $\mathcal{S}_3$. Hence, $MGTP(tr_1(\Pi_1), \mathcal{S}_0) = \mathcal{S}_3$.

5. In $\mathcal{S}_3$, only S_3 satisfies the **T**-condition (9). Hence, $objective(S_3) = \{Q\}$ is the unique stable model of Π_2 by Theorems 3.1 and 3.2.

Note that the above computation is independent of both orders of extensions of model candidates and enumeration of the rules. Furthermore, computation is *incremental*. Consider, for example, that only the first rule is given at first. In this case, $\mathcal{S}_1$ is the output of the MGTP, in which only $\{\neg \mathsf{K}Q, P\}$ satisfies the **T**-condition (9), showing that $\{P\}$ is the stable model of the program. Then, suppose that the second rule is added to the program that contains only the first rule. This time we can see that the MGTP outputs $\mathcal{S}_3$ (so that the stable model is again $\{Q\}$) by using $\mathcal{S}_1$ as the initial model candidates.

Example 3.6 Let the general logic program Π_2 consist of the following four rules:

$$R \leftarrow not\, R\,,$$
$$R \leftarrow Q\,,$$
$$P \leftarrow not\, Q\,,$$
$$Q \leftarrow not\, P\,.$$

These rules are translated to the following MGTP rules:

$$\rightarrow \neg \mathsf{K}R, R \,|\, \mathsf{K}R\,,$$
$$Q \rightarrow R\,,$$
$$\rightarrow \neg \mathsf{K}Q, P \,|\, \mathsf{K}Q\,,$$
$$\rightarrow \neg \mathsf{K}P, Q \,|\, \mathsf{K}P\,.$$

In this example, the first MGTP rule can be further reduced to

$$\rightarrow \mathsf{K}R\,,$$

if we prune the first disjunct by the schema (7). Therefore, the rule has computationally the same effect as the integrity constraint:

$$\leftarrow not\, R\,.$$

This integrity constraint says that every answer set has to contain R: namely, R should be derived. Now, it is easy to see that $MGTP(tr_1(\Pi_2), \{\emptyset\}) = \{ S_5, S_6, S_7 \}$, where $S_5 = \{\mathsf{K}R, \neg \mathsf{K}Q, P, \mathsf{K}P\}$, $S_6 = \{\mathsf{K}R, \mathsf{K}Q, \neg \mathsf{K}P, Q, R\}$, and $S_7 = \{\mathsf{K}R, \mathsf{K}Q, \mathsf{K}P\}$. The only model candidate that satisfies the **T**-condition (9) is S_6, showing that $\{Q, R\}$ is the unique stable model of Π_2. Note that the top-down procedure proposed by Eshghi and Kowalski is not sound [4, pp.251] because it has a top-down proof of P. In our case, we can easily check that $objective(S_5) = \{P\}$ is not a stable model because S_5 contains $\mathsf{K}R$ but does not contain R.

4 Extended Disjunctive Databases

An *extended disjunctive database* [9] is a set of rules of the form:

$$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n \qquad (10)$$

where $n \geq m \geq l \geq 0$ and each L_i is a literal. In particular, when an extended disjunctive database is a set or rules each of whose consequent consists of at most one literal:

$$L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n$$

($n \geq m \geq l \geq 0,\ 1 \geq l \geq 0$), it is called an *extended logic program* [8].

Both extended logic programs and extended disjunctive databases allow for *classical negation* as well as negation as failure. By using these two kinds of negation, we can easily represent incomplete knowledge [8], exceptions [13], closed world assumptions [8, 11], defaults and hypotheses [11].

The answer sets of extended disjunctive databases are defined as generalizations of both the minimal models of positive disjunctive programs and the stable models of general logic programs. The following definition is again based on [10]. Let Π be any extended disjunctive database, and S a subset of $\mathcal{L}$. S is an *answer set* of Π if it is one of the minimal sets of literals S' satisfying the conditions:

1. For any ground instance of any rule of Π

   $$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n \ \ (l \geq 1)\,,$$

   if $L_{l+1}, \ldots, L_m \in S'$ and $L_{m+1}, \ldots, L_n \notin S$, then for some i $(1 \leq i \leq l)$, $L_i \in S'$;

2. For any ground instance of any integrity constraint of Π

   $$\leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n\,,$$

   if $L_1, \ldots, L_m \in S'$ and $L_{m+1}, \ldots, L_n \notin S$, then $S' = \mathcal{L}$;

3. If for some ground atom A, $A \in S'$ and $\neg A \in S'$, then $S' = \mathcal{L}$.

In the same way as general or extended logic programs [11], an extended disjunctive database is classified as either a *consistent*, *contradictory*, or *coherent* database.

We can see that the effect of introducing classical negation into programs appears in the last condition of the above definition. Intuitively speaking,

each answer set is a possible set of beliefs: each literal in an answer set can be considered to be true in the belief set. If neither an atom A nor its negation $\neg A$ is contained in an answer set, the truth value of A is *unknown* in the belief set. Thus the answer set semantics can provide for indefinite answers in answering queries, and such unknown information can be referred to in an extended disjunctive database. In these semantics, positive and negative literals have the same status so that the result of negation by failure to prove A does not mean that $\neg A$ is true. Therefore, unlike positive disjunctive programs, we can no longer interpret extended disjunctive databases as clausal forms of first-order predicate calculus [2].

To compute the answer sets of an extended disjunctive database Π, we translate each rule (10) in Π of the form:

$$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n$$

to the following MGTP rule:

$$L_{l+1}, \ldots, L_m \rightarrow$$

$$\neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_1 \mid \ldots \mid \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_l \mid \mathsf{K} L_{m+1} \mid \ldots \mid \mathsf{K} L_n . \tag{11}$$

The next five schemata are introduced to reject model candidates containing wrong guesses. Let S' be a model candidate.

- If some ground instance of some literal L holds in S' and is not believed in S', then reject S'.

$$\neg \mathsf{K} L, L \rightarrow \quad \text{for every literal } L \in \mathcal{L} . \tag{12}$$

- If some ground instance of some literal L is believed in S' and is not believed in S', then reject S'.

$$\neg \mathsf{K} L, \mathsf{K} L \rightarrow \quad \text{for every literal } L \in \mathcal{L} . \tag{13}$$

- If for some literal L and some substitution σ, S' contains both $L\sigma$ and $\overline{L\sigma}$, then reject S', where $\overline{L\sigma}$ is the literal complementary to $L\sigma$: for instance, when A is an atom, $\overline{A} = \neg A$ and $\overline{\neg A} = A$:

$$L, \overline{L} \rightarrow \quad \text{for every literal } L \in \mathcal{L} . \tag{14}$$

[2] Recall that any set of clauses can be interpreted as a positive disjunctive program [16].

- If for some literal L and some substitution σ, $L\sigma$ holds in S' and $\overline{L\sigma}$ is believed in S', then reject S':

$$\mathsf{K}L,\ \overline{L} \rightarrow \qquad \text{for every literal } L \in \mathcal{L}. \tag{15}$$

- If for some literal L and some substitution σ, both $L\sigma$ and $\overline{L\sigma}$ are believed in S', then reject S':

$$\mathsf{K}L,\ \mathsf{K}\overline{L} \rightarrow \qquad \text{for every literal } L \in \mathcal{L}. \tag{16}$$

We now denote by $tr_2(\Pi)$ the set of rules consisting of the five schemata, (12), (13), (14), (15) and (16), and the MGTP rules obtained by replacing each rule (10) of Π with a rule (11).

Finally, the **T**-condition is as follows. Let be $S' \in MGTP(tr_2(\Pi), \{\emptyset\})$.

- If any ground instance L of any literal is believed in S', then it must be true in S':

$$\text{For every ground literal } L \in \mathcal{L}, \text{ if } \mathsf{K}L \in S', \text{ then } L \in S'. \tag{17}$$

Theorem 4.1 Suppose that Π is consistent. Then,

$$min(\{\ objective(S') \mid S' \in MGTP(tr_2(\Pi), \{\emptyset\}),\\ S' \text{ satisfies the } \mathbf{T}\text{-condition (17)}\ \})$$

is equivalent to the answer sets of Π. □

When the given program Π is not consistent, unlike general logic programs, we cannot identify whether Π is contradictory or incoherent. Here, we have the following weak results.

Proposition 4.2 (1) If Π is contradictory, then $MGTP(tr_2(\Pi), \{\emptyset\}) = \emptyset$. (2) If $MGTP(tr_2(\Pi), \{\emptyset\}) \neq \emptyset$ and there is no model candidate satisfying the **T**-condition (17), then Π is incoherent. □

From the above result, when $MGTP(tr_2(\Pi), \{\emptyset\}) = \emptyset$, we can see that Π is either contradictory or incoherent [3]. Anyway, since Π is inconsistent,

[3]The reason why the MGTP cannot output any model candidate with respect to $tr_2(\Pi)$ for an incoherent database Π is that the fourth and fifth schemata (15), (16) force the MGTP to prune model candidates that may either result in an inconsistent answer set $\mathcal{L}$ or violate the T-condition (17). Therefore, if these two schemata could be removed from the translated program, some model candidates would remain in the final output for the incoherent database. However, such a translation would reduce the efficiency of computation. See also the proof of Theorem 4.1 in the appendix.

if our goal is to compute consistent answer sets, this distinction is not very important. Nevertheless, we could use the following property, which was first given by [11].

Proposition 4.3 Let Π_P be the set of rules in Π not containing *not*. Π is contradictory if and only if Π_P is contradictory. □

Since Π_P is a positive disjunctive program, its answer sets can be easily computed so that we can determine whether the whole database Π is contradictory or not.

Example 4.4 Let us verify Theorem 4.1 in the example of the extended disjunctive database Π_3, which is a slightly modified version of "broken-hand" example of Gelfond et al. [10]:

$$
\begin{array}{l}
\mathit{Lh\text{-}Usable} \leftarrow \mathit{not}\ \mathit{Ab1}\,, \\
\mathit{Rh\text{-}Usable} \leftarrow \mathit{not}\ \mathit{Ab2}\,, \\
\mathit{Ab1} \leftarrow \neg \mathit{Lh\text{-}Usable}\,, \\
\mathit{Ab2} \leftarrow \neg \mathit{Rh\text{-}Usable}\,, \\
\neg \mathit{Lh\text{-}Usable} \mid \neg \mathit{Rh\text{-}Usable} \leftarrow\ .
\end{array}
$$

Of these, the first two rules are translated to the following MGTP rules:

$$
\begin{array}{l}
\rightarrow \neg \mathsf{K}\mathit{Ab1}, \mathit{Lh\text{-}Usable} \mid \mathsf{K}\mathit{Ab1}\,, \\
\rightarrow \neg \mathsf{K}\mathit{Ab2}, \mathit{Rh\text{-}Usable} \mid \mathsf{K}\mathit{Ab2}\,.
\end{array}
$$

It is easy to see that $MGTP(tr_2(\Pi_3), \{\emptyset\})$ contains the following two model candidates that satisfy the **T**-condition (17):

$$
\begin{array}{l}
S_8 = \{\, \neg \mathit{Lh\text{-}Usable},\ \mathit{Ab1},\ \mathsf{K}\mathit{Ab1},\ \neg \mathsf{K}\mathit{Ab2},\ \mathit{Rh\text{-}Usable} \,\}\,, \\
S_9 = \{\, \neg \mathit{Rh\text{-}Usable},\ \mathit{Ab2},\ \mathsf{K}\mathit{Ab2},\ \neg \mathsf{K}\mathit{Ab1},\ \mathit{Lh\text{-}Usable} \,\}\,.
\end{array}
$$

Removing all the K-literals from these model candidates, we can get the two desired answer sets of Π_3.

5 Implementation of Schemata

5.1 Schemata on the MGTP

So far, we have represented schemata to reject model candidates that contain wrong guesses. To implement these schemata, we can simply use object-level schemata on top of the MGTP in the same way as an implementation of tableaux provers of modal logics by [12]. For an atom A, the negative literal $\neg A$ can be expresses as `-A`, and for a literal L, the K-literal $\mathsf{K}L$ can be expressed as `k(L)` in KL1, where neither "-" nor "k" appears elsewhere in

the program as a predicate symbol. The following is an example of expression of the five schemata for extended disjunctive databases.

```
(12)   -k(L), L --> false
(14)   -A, A --> false
(15)   k(A), -A --> false
       k(-A), A --> false
(16)   k(-A), k(A) --> false
```

Note that the schema (13) for extended disjunctive databases can be omitted because it is an instance of the formula (14) above. Also, the two schemata (7) and (8) for general logic programs can be expressed by the above formulas (12) and (14).

By using object-level schemata on the MGTP, we can use the MGTP without any change. This has a great advantage that the inference rules (logic) and the inference engine (control) can be clearly separated. However, to improve the efficiency, we can consider another method as shown next.

5.2 Restriction of Model Candidate Extensions

In Section 2.2, we have defined the model candidate extension operation for a MGTP rule of the form (4). This operation does not distinguish K-literals from objective literals. However, to improve the efficiency, we can incorporate the schemata into the operation so that extensions should be avoided if the resultant model candidates are to be pruned immediately. For example, consider the MGTP rule (11) obtained by translating from a rule (10) in an extended disjunctive database:

$$L_{l+1}, \ldots, L_m \rightarrow$$

$$\neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_1 \mid \ldots \mid \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_l \mid \mathsf{K} L_{m+1} \mid \ldots \mid \mathsf{K} L_n ,$$

where $n \geq m \geq l \geq 0$. For this rule and a model candidate $S' \in \mathcal{S}$, the model candidate extension by the MGTP works as follows:

P1 **if** for some substitution σ, $L_{l+1}\sigma, \ldots, L_m\sigma \in S'$,
P2-1 for any i $(i = 1, \ldots, l)$, it does not hold that
$\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma \in S'$,
P2-2 and for any j $(j = m+1, \ldots, n)$, $\mathsf{K} L_j\sigma \notin S'$,
C1 **then** remove S' from $\mathcal{S}$,
C2-1 for all i $(i = 1, \ldots, l)$,
add $S' \cup \{\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma\}$ to $\mathcal{S}$,
C2-2 and for all j $(j = m+1, \ldots, n)$, add $S' \cup \{\mathsf{K} L_j\sigma\}$ to $\mathcal{S}$.

On the other hand, in a version that incorporates forward-checking of the schemata, the above C2-1 and C2-2 can be replaced with the following:

C2-1′ for all i $(i = 1, \ldots, l)$ such that $\neg \mathsf{K} L_i\sigma, \overline{L_i\sigma}.\mathsf{K}\overline{L_i\sigma} \notin S'$ and that $L_i\sigma \neq L_j\sigma$ for any j $(j = m+1, \ldots, n)$, add $S' \cup \{\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma\}$ to $\mathcal{S}$,

C2-2′ and for all j $(j = m+1, \ldots, n)$ such that $\neg \mathsf{K} L_j\sigma, \overline{L_j\sigma}, \mathsf{K}\overline{L_j\sigma} \notin S'$, add $S' \cup \{\mathsf{K} L_j\sigma\}$ to $\mathcal{S}$.

The five schemata for extended disjunctive databases are completely incorporated into C2-1′ and C2-2′ above. Since this new model candidate extension operation checks whether each new objective or K-literals can be safely added or not to S', all of the schemata shown in the previous subsection is unnecessary [4]. At the expense of these extra checking, we can reduce the number of model candidate extensions. In practice, the cost of generating or extending model candidates and keeping them is much more expensive than the cost of these extra checking. Furthermore, we can dispense with conjunctive matching of antecedents of the schemata, which is always tried against any model candidate even if it is not to be pruned.

6 Discussion

In this section, we compare the proposed method to other approaches to evaluate logic programs containing negation-as-failure formulas.

6.1 Computation

The proposed method has several computational advantages: in a word, it can find *all* answer sets for *every* class of logic program or disjunctive database, *incrementally*, *without backtracking*, and *in parallel*. We shall examine these characteristics as follows:

1. *Finding all answer sets.*
   This fact means that the proposed method is *complete* with respect to the answer set semantics. This is due to the fact that the MGTP [6] can find every minimal model of a positive disjunctive program. For positive disjunctive programs, bottom-up computation has recently been recognized to be more useful than top-down computation, and there has been some other methods to compute minimal models [5, 1] or to characterize fixpoint computation [17]. Therefore, our translation method may be linked with those methods as well as the bottom-up SATCHMO [15] prover. Moreover, our method is *sound* with respect to the answer set semantics, again due to bottom-up computation. Top-down computa-

[4] A forward-checking mechanism will work automatically by using a new version of the MGTP (called the "lazy" MGTP), which does not extend model candidates to be pruned by some integrity constraints.

tion, on the other hand, can never guarantee the soundness even for general logic programs as shown by [4].

2. *Applicable to every class of logic program or deductive database.*
   Several procedures have been proposed to compute the stable models of general logic programs [19, 3, 21], but none of them can be extended to allow for disjunctive databases because they are based on TMS-like algorithms. Note also that our proposed method does not increase the computational complexity of the problem more than computation of the minimal models of positive disjunctive programs; the size of the translated MGTP rules is the same as the size of the original rules, and the disjuncts introduced by the translation would be of the same size of the positive literals in the heads of the positive disjunctive database if each negation-as-failure formula *not A* were replaced with a negative literal $\neg A$ in the sense of classical clausal logic.

3. *Incremental, backtrack-free computation.*
   Since we keep K-literals in each model candidate, when new rules are added to the database, the previous set of model candidates can be used as the input to the next computation. Procedures given by [19, 21] cannot be used incrementally. Eshghi's [3] proposal may be used incrementally, but it requires an exponential-time algorithm to convert its data structures into the stable models, which is much more complicated than our use of the T-condition (9). Furthermore, by means of case-splitting of the MGTP, a model candidate is split into multiple model candidates, without future backtracking. We thus need not enumerate rules for their applications to model candidate extensions.

4. *Parallel implementation.*
   Our method is also the first attempt to compute answer sets in parallel. The procedure has been implemented on a distributed-memory multiprocessor machine, Multi-PSI, developed in ICOT. The transformation is especially suitable for OR-parallelism because for each negation-as-failure formula we will make guesses to believe or disbelieve it. Multiple model candidates are thus taken as the source for exploiting OR-parallelism of the MGTP.

6.2 Application to Legal Reasoning

The proposed method is currently being applied to a legal reasoning system developed at ICOT. We can see some advantages of the proposed method

from the viewpoint of this application.

It has been recognized that to use two kinds of negation, negation as failure and classical negation, are very powerful in order to represent knowledge of legislation in logic programs [13, 20]. In [20], a *primary* fact (whose proof has to be demonstrated) can be represented by a literal, while a *secondary* fact L (for which proof to the contrary must not be given) as a negation-as-failure formula $not\ \overline{L}$. The question is how we should use logic programming in legal reasoning. Usually, in a legal reasoning system, a set of ground facts and a set or general rules or norms are given, and the goal is to obtain the possible *interpretations* containing judicial precedents. However, such an inference is plausible as it is often necessary to reason with incomplete information. Therefore, the system should create explanations or justifications why the conclusions have been derived under some legal concept. This kind of processes sometimes reflects antagonistic arguments by a jury in a court.

For the above purposes, our computation is extremely desirable. Bottom-up computation constructs the model candidates, each of which corresponds to a possible interpretation. In each model candidate, a negative K-literal $\neg \mathsf{K}\, L$ represents an absence of the contrary to a secondary fact $\overline{L}$. Thus, if a proof for L could be given against $\neg \mathsf{K}\, L$, then the corresponding argument would be rebuted. On the other hand, since the objective literals in a model candidate that does not satisfy the T-condition is not an answer set, the model candidate can be understood as a weak argument. In this case, though, we can see that if only a literal L could be established for each positive K-literal $\mathsf{K}\, L$ that has not been justified in the model candidate, such an argument might become valid. Hence, those extra information represented by K-literals in model candidates can play an important role in legal reasoning.

7 Conclusion

In this paper, we have presented a novel technique to compute answer sets of logic programs or disjunctive databases. The technique is simply based on a bottom-up model generation method for positive disjunctive databases, together with integrity constraints over K-literals expressed by object-level schemata on the MGTP. The proposed transformation is also very simple and does not increase the program size. Moreover, the method has been implemented on a parallel inference machine.

We should comment, though, that while our results have a useful application to legal reasoning, the general question of how to avoid combinatorial explosion in constructing model candidates still needs to be investigated. From our experience in testing our procedure for some kinds of nonmonotonic

reasoning and planning, it is necessary for some applications to have a query-answering mechanism in addition to computation of the model candidates. We also expect that for different semantics of logic programs or disjunctive databases from answer set semantics, it may be possible to have different transformations into MGTP rules containing K-literals together with different integrity constraints. These issues will be discussed in a separate paper.

Acknowledgment

We were motivated to work on this topic by the request of the legal reasoning group in ICOT. We are grateful to Katsumi Nitta and Hiroshi Ohsaki for their cooperation.

References

[1] C. Bell, A. Nerode, R.T. Ng and V.S. Subrahmanian, Implementing deductive databases by linear programming, Technical Report UMIACS-TR-91-122, Department of Computer Science, University of Maryland, College Park, MD, August 1991.

[2] G. Bossu and P. Siegel, Saturation, nonmonotonic reasoning, and the closed-world assumption, *Artificial Intelligence* **25** (1985) 23–67.

[3] K. Eshghi, Computing stable models by using the ATMS, in: *Proceedings of AAAI-90*, Boston, MA (1990) 272–277.

[4] K. Eshghi and R.A. Kowalski, Abduction compared with negation by failure, in: *Proceedings of the Sixth International Conference on Logic Programming*, Lisbon, Portugal (1989) 234–254.

[5] J.A. Fernández and J. Minker, Bottom-up evaluation of hierarchical disjunctive deductive databases, in: *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France (1991) 660–675.

[6] H. Fujita and R. Hasegawa, A model generation theorem prover in KL1 using a ramified-stack algorithm, in: *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France (1991) 535–548.

[7] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, Seattle, WA (1988) 1070–1080.

[8] M. Gelfond and V. Lifschitz, Logic programs with classical negation, in: *Proceedings of the Seventh International Conference on Logic Programming*, Jerusalem, Israel (1990) 579–597.

[9] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, *New Generation Computing* (1991) to appear.

[10] M. Gelfond, V. Lifschitz, H. Przymusińska and M. Truszczyński, Dis-

junctive defaults, in: *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1991) 230–237.

[11] K. Inoue, Extended logic programs with default assumptions, in: *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France (1991) 490–504.

[12] M. Koshimura and R. Hasegawa, Modal propositional tableaux in a model generation theorem prover, in: *Proceedings of the Logic Programming Conference '91*, Tokyo, Japan (1991) 43–52 (in Japanese).

[13] R.A. Kowalski and F. Sadri, Logic programs with exceptions, in: *Proceedings of the Seventh International Conference on Logic Programming*, Jerusalem, Israel (1990) 598–613.

[14] V. Lifschitz, Nonmonotonic databases and epistemic queries, in: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia (1991) 381–386.

[15] R. Manthey and F. Bry, SATCHMO: a theorem prover implemented in Prolog, in: *Proceedings of the Ninth International Conference on Automated Deduction*, Lecture Notes in Computer Science **301**, Springer-Verlag (1988).

[16] J. Minker, On indefinite databases and the closed world assumption, in: *Proceedings of the Sixth International Conference on Automated Deduction*, Lecture Notes in Computer Science **138**, Springer-Verlag (1982) 292–308.

[17] D.W. Reed, D.W. Loveland and B.T. Smith, An alternative characterization of disjunctive logic programs, Technical Report CS-1991-11, Department of Computer Science, Duke University, Durham, NC, July 1991.

[18] R. Reiter, A logic for default reasoning, *Artificial Intelligence* **13** (1980) 81–132.

[19] D. Saccà and C. Zaniolo, Stable models and non-determinism in logic programs with negation, in: *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, TN (1990) 205–229.

[20] G. Sartor, The structure of norm conditions and nonmonotonic reasoning in law, in: *Proceedings of the Third International Conference on Artificial Intelligence and Law*, Oxford, England (1991) 155–164.

[21] K. Satoh and N. Iwayama, Computing abduction by using the TMS, in: *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France (1991) 505–518.

A Appendix: Proof of Theorem 4.1

Here, we give a sketch of the proof of Theorem 4.1. Since Theorems 3.1 and 3.2 are instances of Theorem 4.1, these proofs are omitted. We will also omit proofs of other theorems and propositions as they can be proved more easily. In the following, we assume without loss of generality that Π is a finitely groundable extended disjunctive database. Therefore, we can further assume that Π is a set of ground rule of the form (10). We use the following notations throughout this appendix. If R is a rule in Π of the form (10):

$$R \quad = \quad L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n\,,$$

we define the following:

$$\begin{aligned}
tr(R) &= L_{l+1}, \ldots, L_m \rightarrow \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_1 \mid \ldots \\
&\qquad \ldots \mid \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_l \mid \mathsf{K} L_{m+1} \mid \ldots \mid \mathsf{K} L_n\,, \\
red(R) &= L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m\,, \\
dis(R) &= \{L_1, \ldots, L_l\}\,, \\
pos(R) &= \{L_{l+1}, \ldots, L_m\}\,, \\
neg(R) &= \{L_{m+1}, \ldots, L_n\}\,, \\
\mathsf{K}neg(R) &= \{\mathsf{K} L_{m+1}, \ldots, \mathsf{K} L_n\}\,, \\
\neg\mathsf{K}neg(R) &= \{\neg\mathsf{K} L_{m+1}, \ldots, \neg\mathsf{K} L_n\}\,.
\end{aligned}$$

Given Π and a set of literals $S \subseteq \mathcal{L}$, we define:

$$\mathit{reduct}(\Pi, S) = \{\ \mathit{red}(R) \mid R \in \Pi,\ \mathit{neg}(R) \cap S = \emptyset\ \}\,.$$

The next is a well-known property of answer sets.

Lemma A.1 S is an answer set of Π if and only if S is an answer set of $\mathit{reduct}(\Pi, S)$. □

The next two lemmas are the main parts in the proof of Theorem 4.1.

Lemma A.2 Let be $S' \in MGTP(tr_2(\Pi), \{\emptyset\})$. If S' satisfies the **T**-condition (17), then $S = \mathit{objective}(S')$ is contained in $MGTP(\mathit{reduct}(\Pi, S), \{\emptyset\})$.

Proof: For every rule $tr(R)$ in $tr_2(\Pi)$, it holds that, if $\mathit{pos}(R) \subseteq S'$, then either $\mathsf{K}neg(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $\neg\mathsf{K}neg(R) \subseteq S'$. By the schema (12) and the **T**-condition (17), for every rule $tr(R)$ in $tr_2(\Pi)$, if $\mathit{pos}(R) \subseteq S'$, then either $\mathit{neg}(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $\mathit{neg}(R) \cap S' = \emptyset$. Now,

for each rule $tr(R)$ in $tr_2(\Pi)$, there is a rule R in Π. Therefore, for every rule R in Π, if $pos(R) \subseteq S$, then either $neg(R) \cap S \neq \emptyset$, or $dis(R) \cap S \neq \emptyset$ and $neg(R) \cap S = \emptyset$. Because of the closedness of the MGTP operations, this S is contained in $MGTP(reduct(\Pi, S), \{\emptyset\})$. □

Note that by the above proof, only the schema (12) and the **T**-condition (17) are necessary to guarantee the soundness of computation; other schemata are used for obtaining the efficiency of computation.

Lemma A.3 Let be $S \in MGTP(reduct(\Pi, S), \{\emptyset\})$. If Π is consistent, then there is a model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that $S = objective(S')$ and S' satisfies the **T**-condition (17).

Proof: For every rule R in $reduct(\Pi, S)$, it holds that, if $pos(R) \subseteq S$, then $dis(R) \cap S \neq \emptyset$. We then see that for every rule R in Π, if $pos(R) \subseteq S$, then either $neg(R) \cap S \neq \emptyset$, or $dis(R) \cap S \neq \emptyset$ and $neg(R) \cap S = \emptyset$. Now, for each rule R in Π, there is a rule $tr(R)$ in $tr_2(\Pi)$. Since $MGTP(reduct(\Pi, S), \{\emptyset\})$ is closed under the operations of the MGTP, there exists a model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that $S = objective(S')$ and that for any rule $tr(R)$ in $tr_2(\Pi)$, it holds that, if $pos(R) \subseteq S'$, then either $\mathsf{K}neg(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $\neg\mathsf{K}neg(R) \subseteq S'$. It is easy to check that this S' is not pruned by the schemata, and that S' satisfies the **T**-condition (17). □

Now, we prove the Theorem.

Theorem 4.1 Suppose that Π is consistent. Then,

$$min(\{\, objective(S') \mid S' \in MGTP(tr_2(\Pi), \{\emptyset\}),\\ S' \text{ satisfies the } \mathbf{T}\text{-condition (17)} \,\})$$

is equivalent to the answer sets of Π.

Proof: A set S of literals is an answer set of Π if and only if S is an answer set of $reduct(\Pi, S)$ (by Lemma A.1) if and only if

$$S \in min(MGTP(reduct(\Pi, S), \{\emptyset\}))$$

by Proposition 2.1. By Lemma A.3, if S is contained in the above set of model candidates, then there is a model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that S' satisfies the **T**-condition (17), and that $S = objective(S')$ is a minimal set satisfying these conditions. Conversely, by Lemma A.2, if a model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ satisfies the **T**-condition (17), and $S = objective(S')$ is such a minimal set, then S is contained in $min(MGTP(reduct(\Pi, S), \{\emptyset\}))$. Hence, the theorem holds. □