Computing Stable Models By Using the ATMS

Kave Eshghi
Hewlett Packard Laboratories,
Filton Road,
Stoke Gifford,
Bristol BS12 6QZ,
England.
Email:ke@hplb.hpl.hp.com

Abstract

An algorithm is described which computes stable models of propositional logic programs with negation as failure using the Assumption Based Truth Maintenance mechanism. Since stable models of logic programs are closely connected to stable expansions of a class of auto-epistemic theories, this algorithm points to a link between stable expansions of a class of auto-epistemic theories and ATMS structures

Introduction

In this paper an algorithm is described which computes stable models of propositional logic programs with negation as failure [1], using the Assumption Based Truth Maintenance mechanism [2]. Since stable models of logic programs are closely connected to stable expansions of a class of auto-epistemic theories, this algorithm points to a link between stable expansions of a class of auto-epistemic theories and ATMS structures.

Stable Models of Logic Programs

Stable models of logic programs with negation as failure were introduced in [1] as a means of specifying the semantics of logic programs. They are defined as follows:

A logic program is a set of clauses of the type

$$p \leftarrow p_1, p_2, \ldots \sim q_1, \sim q_2 ..$$

where $\sim$ indicates negation as failure. In this paper we only consider propositional logic programs, where $p_1, p_2, ..q_1, q_2 ..$ are propositions. We do not place any other restriction on the structure of the clasues.

Definition 1 *The answer set of a propositional Horn Clause program is the set of all propositions provable from it. We use Answer(*Π*) to denote the answer set of the program* Π

Definition 2 *Let P be a propositional logic program with negation as failure, and I a set of propositions. The negation-free program* P_I *is derived from P by*

- *Deleting all the clauses in P which have a negative conditions such as* $\sim q$ *where* $q \in I$
- *Deleting all the negative conditions in all the remaining clauses of P.*

I is a stable model of P iff $I = Answer(P_I)$

Example: let P be the program

$$a \leftarrow\sim b$$

$$b \leftarrow\sim c$$

Then $P_{\{b\}} = \{b \leftarrow\}$, and therefore $\{b\}$ is a stable model of P.

In general, a logic program may have any number of stable models. The algorithm described here computes all of them.

Stable models of logic programs are closely connected to stable expansions of auto-epistemic theories. In fact, as observed in [1], if we replace every negative condition $\sim p$ in the program P with the condition $\neg B(p)$, where B is the belief operator of auto-epistemic logic, the stable expansion of the resulting auto-epistemic theory is the same as the stable model of P. As such, the algorithm described in this paper can be considered to be a theorem prover for a restricted class of auto-epistemic theories.

The ATMS

We are only concerned with a subset of the capabilities of the ATMS, which we define below.

Let N be a set of propositions, called *nodes*, and S a distinguished subset of N, called *assumptions*. Let J be a set of propositional Horn Clauses, called justifications, in which all propositional symbols are from N. When we transmit S and J to the ATMS, it computes the following structures:

- All the minimal sets of assumptions ϵ such tht $J \cup \epsilon$ is inconsistent. These sets arc called nogoods.
- for every node n, all the minimal sets of assumptions e such that $J \cup e \vdash n$. e is called an *environment* of n, and the set of all environments of a node n is called the *label* of n.

For example, let J be

$a \leftarrow b^*$

$\leftarrow b, b^*$

$b \leftarrow c^*$

$\leftarrow c, c^*$

and $S = \{b^*, c^*\}$. Then the ATMS will compute the only nogood to be $\{b^*, c^*\}$, and the following labels for each node:

$a : \{\{b^*\}\}$

$b : \{\{c^*\}\}$

$c : \{\}$

(Note: $\leftarrow c, c^*$ is an alternative syntax for $\neg(c \wedge c^*)$, where $\neg$ signifies classical negation)

Notation: we will use $nogood^{J,S}(\epsilon)$ to indicate that ϵ is a nogood given the justifications J and assumptions S. We will use $env^{J,S}(n, e)$ to indicate that e is an environment of n given justifications J and assumptions S.

The Algorithm

To compute the stable models of a logic program P, we proceed as follows:

1. From the program P derive the set of Horn clauses P^* and the set of assumptions S;
2. Add the elements of S to the ATMS as assumptions, and add the clauses in P^* to the ATMS as justifications;
3. From the data structures computed by the ATMS (environments and nogoods) compute the stable model of P.

Each one of these steps is described below.

Deriving P^* and S

Given a propositional logic program P, P^* and S are derived as follows: for every clause such as

$$p \leftarrow p_1, p_2, .. \sim q_1, \sim q_2 ... \sim q_n$$

in P, add the assumptions $q_1^*, q_2^*, ...q_n^*$ to S, where $q_1^*, q_2^* ... q_n^*$ are propositional symbols not occuring in P, and add the Horn clauses

$p \leftarrow p_2, p_2.., q_1^*, q_2^* ... q_n^*$

$\leftarrow q_1, q_1^*$

$\leftarrow q_2, q_2^*$

....

$\leftarrow q_n, q_n^*$

to P^*. (We call q and q^* the *complements* of each other. In the rest of this paper, an assumption indicated as q^* will always be the complement of the propositional symbol q).

For example, let P be

$a \leftarrow\sim b$

$b \leftarrow\sim c$

Then P^* is

$a \leftarrow b^*$

$\leftarrow b, b^*$

$b \leftarrow c^*$

$\leftarrow c, c^*$

and $S = \{b^*, c^*\}$.

The relationship between P and P^*

As mentioned before, the stable models of P are related to the environments and nogoods returned by the ATMS when it is presented with the assumptions in S and the justifications in P^*. To explore this relationship, first we define the notion of *stable generator.*

Definition 3 *A set of assumptions σ is a stable generator of P iff*

$P^* \cup \sigma$ *is consistent*

$\forall n^* \in S, (n^* \notin \sigma \rightarrow P^* \cup \sigma \vdash n)$

(Note: n is the complement of n^*)

The following two theorems establish the link between stable generators and stable models:

Theorem 1 *Let σ be a stable generator of P. Then the set*

$$I = Answer(P^* \cup \sigma) \setminus \sigma$$

is a stable model of P.

Theorem 2 *Let I be a stable model of P. Then the set*

$$\sigma = \{q^* : q^* \in S \wedge q \notin I\}$$

is a stable generator of P.

From these theorems it follows that for every stable model of P, there is a unique stable generator. Furthermore, given a stable generator, it is straightforward to compute the corresponding stable model. The algorithm described in this paper computes stable generators of a progam P from the environments and nogoods computed by the ATMS when it is given the assumptions S and justifications P^*. To do this, we need the following lemma.

Lemma 1 *A set of assumptions σ is a stable generator of P iff*

$\forall \epsilon, nogood^{P^*,S}(\epsilon) \rightarrow \epsilon \not\subset \sigma$

$\forall n^* \in S, (n^* \notin \sigma \rightarrow \exists e, (env^{P^*,S}(n, e) \wedge e \subset \sigma))$

(Note: n is the complement of n^*)

This lemma directly follows from the definiton of stable generators and the properties of ATMS nogoods and environments.

Example: When the S and P^* computed in example 1 are transmitted to the ATMS as assumptions and justifications, $\{b^*, c^*\}$ will be the only nogood. The nodes in the ATMS, with their labels, will be:

$a : \{\{b^*\}\}$

$b : \{\{c^*\}\}$

$c : \{\}$

In this situation, $\{c^*\}$ is the only set of assumptions satisfying the definition of a stable generator, i.e. it is the generator of the only stable model of P. $\{b^*\}$ is not a stable generator because it does not safisfy the second requirement of the definition of stable generators, for $c^* \notin \{b^*\}$ but $P^* \cup \{b^*\} \not\vdash c$.

Computing Stable Generators

In this section, an algorithm is described which, given the environments and nogoods returned by the ATMS, computes all stable generators of P. The algorithm computes a stable generator by finding a set of assumptions, τ, such that $S \setminus \tau$ is a stable generator. (We call τ an acceptable culprit set if $S \setminus \tau$ is a stable generator)

The intuition behind the algorithm is as follows: find a set of assumptions τ such that by removing them from S, the remaining set of assumptions σ satisfies the requirements of lemma 1. The first requirement, that no nogood set should be a subset of σ, can be satisfied by choosing one assumption from each nogood and including them in τ. (This is similar to computing diagnoses from a set of conflict sets [9][3]). However, if we remove an assumption q^* from S where q does not have any environment which is a subset of the remaining set of assumptions we violate the second requirement of lemma 1. For example, consider the only nogood set for the example program above, $\{b^*, c^*\}$. If we choose b^* as the culprit, we will satisfy the requirements of lemma 1, because b has an environment, $\{c^*\}$, which is a subset of the remaining set of assumptions, $\{c^*\}$. But if we choose c^* as the culprit, we violate the second requirement of lemma 1, because c does not have any environment which is a subset of the remaining set of assumptions $\{b^*\}$.(In fact, c does not have any environments at all).

We call the environment e a complement environment of the assumption q^* if it is an environment of q. We call the set of all complement environments of q^* the complement label of q^*. In order to make sure that the second requirement of lemma 1 is satified, the algorithm keeps with each chosen culprit q^* the set of all its complement environments which do not include any of the other culprits. We call this set of complement environments the companion of assumption q^*. Every time a new culprit is chosen, the companions of all the other culprits are updated to remove the environments which include the chosen culprit. If we can choose a culprit from each nogood without making the companion of any other culprit empty, this will be an acceptable culprit set, because the remaining set of assumptions will satisfy lemma 1, and will therefore be a stable generator. The details of this process are given below.

Data Structures and Procedures

The data structure used to keep assumptions together with their companions is the augmented assumption. An augmented assumption is a tuple (q^*, E) where q^* is an assumption, and E is a list of complement environments of q^*. We call q^* the main assumption of (q^*, E).

The list of augmented assumptions

$$[(n_1^*, E_1), (n_2^*, E_2)..(n_k^*, E_k)]$$

is an augmented nogood when its main assumption set, i.e. $\{n_1^*, n_2^*....n_k^*\}$, is a nogood (as returned by the ATMS) and for all i, E_i is the complement label of n_i^* (i.e. the list of *all* complement environments of n_i^*). The augmented nogood list relating to example 1 is

$$[(b^*, [[c^*]]), (c^*, [])]$$

At the heart of the algorithm is the procedure stablecomp. If A is the list of all augmented nogoods of P^*, stablecomp(A, Tau) succeeds iff Tau is a set of acceptable culprits (i.e. $S \setminus Tau$ is a stable generator of P).

We have presented the algorithm in Prolog because it is a non-deterministic algorithm. Prolog handles the non-determinism by its built-in backtracking.

Below is the top level definition of stablecomp.

```
stablecomp(A,Tau):-
  stcomp(A,[],Comp),
  main_assumptions(Comp,Tau).
```

stablecomp is defined in terms of stcomp, which recursively chooses one assumption from each nogood as the next culprit, maintaining the list of culprits chosen so far with their companions in the argument Culprits_sofar.

```
stcomp([],X,X).
stcomp([Nogood|Nogoods],Culprits_sofar,Comp):-
  choose(Nogood,(Node*,Label)),
  filter1(Culprits_sofar,Label,Label1),
  filter2(Node*,Culprits_sofar,Culprits1),
  filter3(Node*,Nogoods,Nogoods1),
  stcomp(Nogoods1,[(Node*,Label1|Culprits1],Comp).
```

To explain the behaviour of stcomp, first we describe the data structures in its argument places, then we describe the recursion invariant of the above clauses, and

then the constituent predicats choose, filter1,filter2 and filter3.

- The first argument of stcomp is a list of augmented nogoods.
- Its second argument is a list of augmented assumptions $[(a_1^*, E_1), (a_2^*, E_2)...]$, where a_i^* are the chosen assumptions sofar, and E_i are their companions.
- The third argument of stcomp is the variable that is used to return the result of the computation

The following is the recursion invariant for the definition of stcomp. It holds when stcomp is initially called by stablecomp, and whenever it is subsequently called in the recursion.

Invariant 1
Let τ be the main assumptions set $\{a_1^*, a_2^*, ..\}$ of Culprits_sofar. *Then the following will always hold whenever* stcomp(Augmented_nogoods,Culprits_sofar,Comp) *is called.*

1. *For every augmented assumption (n_i^*, E_i) in* Culprits_sofar, *E_i is not empty, and it is the list of all complement environments of n_i^* whose intersection with τ is empty,*
2. Augmented_Nogoods *is a list of all augmented nogoods of P^* the intersection of whose main assumption set with τ is empty.*

These conditions are clearly satisfied when stcomp is initially invoked. Let τ be the set of main assumptions of Culprits_so_far when the base case of recursion is reached. Then it will be the case that

- Every assumption a^* in τ will have at least one complement environment whose intersection with τ is empty,
- Every nogood will have at least one element from *tau*

Thus τ will be an acceptable culprit set, and $S \setminus \tau$ will be a stable generator.

The constituent procedures of stcomp

Below we discuss the function of the choice procedure and the three filter procedures.

choose The call choose(Nogood,(Node*,Label)) nondeterministically chooses an augmented assumption (Node*,Label) from the augmented nogood Nogood.

filter1 Let τ be the main assumption set of Culprits_sofar. The function of the filter1 procedure is to remove from the complement-label of the chosen assumption all the environments which include an assumption which occurs on τ. Thus, if

filter1(Culprits_sofar,Label,Label1)

is successful, Label1 will be the list of all the environments in Label whose intersection with τ is empty. If all the environments in Label include an assumption from τ, then filter1(Culprits_sofar,Label,Label1) fails.

filter2 The function of this procedure is to remove from the companions of all assumptions on Culprits_sofar those environments which include the chosen assumption. Let Culprits_sofar be the list $[(n_1^*, E_1), (n_2^*, E_2)..]$. Then if

filter2(Node*,Culprits_sofar,Culprits1)

is successful, Culprits1 will be the list $[(n_1^*, E_1'), (n_2^*, E_2')...]$ where E_i' is the subset of E_i with all the environments in E_i which include Node* removed. If there is an E_i which would be empty after removing all environments which contain Node*, filter2 fails.

filter3 Let Nogoods be a list of augmented nogoods. Then the procedure call

filter3(Node*,Nogoods,Nogoods1)

removes from Nogoods all the augmented nogoods whose main assumption set includes Node*, returning the result in Nogoods1.

Generating all stable generators

The procedure stablecomp described above returns one acceptable culprit set when invoked, from which the corresponding stable generator can be computed by the set complement operation. To find all stable generators, the Prolog findall procedure can be used to find all the acceptable culprit sets.

Related work

The relationship between truth maintenance and nonmonotonic reasoning formalisms is close, and there is an extensive literature on it. Of direct relevance to us is the work reported in [7], where an algorithm (different from the one presented here) is presented for computing stable models of propositional logic programs. The major difference between our work and the one reported in [7] is that they compute stable models directly, and advocate their use as an *alternative* to the ATMS, whereas the work presented here derives the stable models from ATMS structures, and sheds some light on the relationship between stable expansions of auto-epistemic theories and ATMS structures.

In [6] and [8] the semantics of Doyle's TMS [4] are related to auto-epistemic expansions. Their work can be considered complimentary to ours, because they start from a given truth maintenance system (Doyle's TMS) and give its semantics in terms of auto-epistemic expansions, whereas we start from a class of auto-epistemic theories and relate their expansions to ATMS structures.

The two theorems stated above are a special case of a more general theorem proved in [5], in which a link was established between stable models of logic programs and abductive hypotheses generation.

References

[1] Gelfond,M. & Lifschitz,V: "The stable model semantics for logic programming", Proceeding Fifth International Conference on Logic Programming, MIT Press 1988

[2] DeKleer, J: "An assumption based truth maintenance system", Artificial Intelligence Journal 28, 1986

[3] DeKleer, J. & Williams, B.C.: "Diagnosing multiple faults", Artificial Intelligence Journal 32, 1987

[4] Doyle, J: "A truth maintenance system", Artificial Intelligence Journal 12, 1979

[5] Eshghi,K & Kowalski, R.A. "Abduction compared with negation as failure", Proceedings of the Sixth International Conference on Logic Programming, MIT Press 1989

[6] Fujiwara, Y & Honiden, S: "Relating the TMS to auto-epistemic logic", Proceedings IJCAI 89, Morgan Kaufman inc., 1989

[7] Pimental, S.G. & Cuadraro, J.L.: "A truth maintenance system based on stable models", Proceedings of the North American Conference on Logic Programming, MIT Press 1989

[8] Reinfrank,M. et.al.: "On the relation between truth maintenance and auto-epistemic logic", Proceedings IJCAI 89, Morgan Kaufman inc., 1989

[9] Reiter,R: "A theory of diagnosis from first principle", Artificial Intelligence Journal 32, 1987

Appendix

In this appendix the two theorems are proved. First, a few preliminaries.

Definition 4 *Let P be a propositional logic program. Let P^* and S be derived as described in the paper. Let σ be a subset of S. Then P^*_σ is the horn clause program derived from P^* by:*

1. *Removing every clause which has a condition q^* where $q^* \in S \setminus \sigma$*
2. *From the remaining clauses in P^*, removing every condition q^* where $q^* \in \sigma$*

Lemma 2 *For a program P^*_σ defined as above,*

$$Answer(P^*_\sigma) = Answer(P^* \cup \sigma) \setminus \sigma$$

Proof Easily follows from the definition of P^* and P^*_σ

Proof of Theorem 1

From the definiton of I in theorem 1, the definition of P^*_σ and *lemma*2 it follows that

$$I = Answer(P^*_\sigma)$$

Below we prove that $P^*_\sigma = P_I$, thus proving that

$$I = Answer(P_I)$$

which proves that I is a stable model of P.

Proof of $P_I = P^*_\sigma$

1. Let $c = p \leftarrow p_1, p_2, ...$ be a clause in P_I. If c occurs in P, since it does not have any negative conditions it will be in P^*_σ as well.

   Otherwise, there is a clause

   $$p \leftarrow p_1, p_2, ..., \sim q_1, \sim q_2 ...$$

   in P where $q_1, q_2 ... \notin I$.

   Let q_n be any of $q_1, q_2 ....$ Since $\sim q_n$ occurs as a condition of a clause in P, $q^*_n \in S^*$. Since σ is a stable generator of P, $q^*_n \in \sigma \leftarrow P^* \cup \sigma \not\vdash q_n$. But since $q_n \notin I$, by definition of I $P^* \cup \sigma \not\vdash q_n$ which shows that $q^*_n \in \sigma$.

   Now, since $p \leftarrow p_1, p_2, ..., \sim q_1, \sim q_2 ...$ is in P, by definition of P^* there is a clause $p \leftarrow p_1, p_2, ...q^*_1, q^*_2 ..$ in P^*. Since, as proved above, $q^*_1, q^*_2, .. \in \sigma$, from the definition of P^*_σ it follows that $p \leftarrow p_1, p_2 ...$ is in P^*_σ, i.e. c is in P^*_σ

2. Let $c = p \leftarrow p_1, p_2 ..$ be a clause in P^*_σ. If c occurs in P, since it does not have any negative conditions it will be in P_I as well.

   Otherwise, there is a clause $p \leftarrow p_1, p_2, ...q^*_1, q^*_2 ..$ in P^* where $q^*_1, q^*_2 .. \in \sigma$. Now, let q^*_n be any of $q^*_1, q^*_2 ...$ Since σ is a stable generator of P, $P^* \cup \sigma$ is consistent. Furthermore, by definition P^* contains a clause $\leftarrow q_n, q^*_n$. Thus $P^* \cup \sigma \not\vdash q_n$, for otherwise $P^* \cup \sigma$ would be inconsistent. Thus, by definiton of I, $q_n \notin I$.

   Now, since there is a clause $p \leftarrow p_1, p_2, ...q^*_1, q^*_2 ..$ in P^*, there is a clause $p_1, p_2, ... \sim q_1, q_2 ..$ in P, where as shown above, $q_1, q_2, ... \notin I$. Thus $p \leftarrow p_1, p_2, ...$ belongs to P_I, i.e. $c \in P_I$.

Proof of Theorem 2

First, we prove that $P^*_\sigma = P_I$

1. Let $c = p \leftarrow p_1, p_2 ..$ be a clause in P_σ.

   If c occurs in P, since it does not have any negative conditions it will be in P_I as well.

   Otherwise, there is a clause

   $$p \leftarrow p_1, p_2, ..q^*_1, q^*_2 ..$$

   in P^* where $q^*_1, q^*_2, .. \in \sigma$. Let q^*_n be any of $q^*_1, q^*_2 ....$ Since $q^*_n \in \sigma$, by definiton of σ, $q_n \notin I$.

   Since

   $$p \leftarrow p_1, p_2, ..q^*_1, q^*_2 ..$$

   is in P^*, there is a clause

   $$p \leftarrow p_1, p_2, .. \sim q_1, \sim q_2 ..$$

   in P where $q_1, q_2 ... \notin I$. Thus $p \leftarrow p_1, p_2 ..$ is in P_I, i.e. $c \in P_I$.

2. Let $c = p \leftarrow p_1, p_2 ..$ be a clause in P_I. Then there is a clause

   $$p \leftarrow p_1, p_2, .. \sim q_1, \sim q_2 ..$$

   in P, where $q_1, q_2, .. \notin I$. Let q_n be any of $q_1, q_2 ....$ Since $\sim q_n$ occurs as a condition in P, $q^*_n \in S$. Thus, since $q_n \notin I$, by definiton of σ, $q^*_n \in \sigma$.

Since

$$p \leftarrow p_1, p_2, .. \sim q_1, \sim q_2 ..$$

is in P, there is a clause

$$p \leftarrow p_1, p_2, .. q_1^*, q_2^* ..$$

in P^* where $q_1^*, q_2^* ... \in \sigma$. Thus $p \leftarrow p_1, p_2 ..$ is in P_σ, i.e. $c \in P_\sigma$.

Now we show σ is a stable generator by showing that it satisfies the two requirements of stable generators.

1. We prove that $P^* \cup \sigma$ is consistent by contradiction. Suppose $P^* \cup \sigma$ is inconsistent. Then, the proof of inconsistency would involve a denial of the form $\leftarrow p, p^*$, where $P^* \cup \sigma \vdash p$ and $P^* \cup \sigma \vdash p^*$. But since, due to the definiton of P^*, assumptions do not occur at the head of clauses, the only way p^* can be provable from $P^* \cup \sigma$ is for it to be a member of σ. But, due to the definiton of σ, this means that $p \notin I$, which means that $P_I \nvdash p$, i.e, since $P_I = P_\sigma^*$, $P_\sigma^* \nvdash p$. Thus, by definiton of P_σ^*, $P^* \cup \sigma \nvdash p$, leading to a contradiction.
2. To prove that $\forall n^* \in S, (n^* \notin \sigma \rightarrow P^* \cup \sigma \vdash n)$, assume that $n^* \in S$ and $n^* \notin \sigma$. Then by definition of σ, $n \in I$, which, since I is a stable model of P, means that $P_I \vdash n$. Since $P_I = P_\sigma^*$, this means that $P_\sigma^* \vdash n$, i.e. that $P^* \cup \sigma \vdash n$