TR-0803

Magic Set 法による安定モデル上の General Logic Program インタープリタ Draft

藤田　正幸、岩山　登、長谷川　隆三

September, 1992

ICOT

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

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Magic Set 法による安定モデル上の General Logic Program インタープリタ
Draft

藤田 正幸
岩山 登
長谷川 隆三
(財) 新世代コンピュータ技術開発機構

1992 年 8 月 6 日

概要

安定モデルの意味論をもつ General Logic Program のインタプリタを Prolog をメタインタプリタとして実現した。ここで述べるインタプリタは、Prolog をメタインタプリタとしたことにより、質問応答の機能をもち、同時に、質問応答問題ゆえ、解が含まれている安定モデルのみを求める手続きであるという点で、効率がよいものとなっている。また、データベースの Magic Set 法を応用し、トップダウンの制御によるボトムアップ手続きを導入することにより、あるクラスの問題に対しては完全な Q/A 手続きになっている。また、インタプリタは 1 ページ以下の Prolog プログラムである。

Data Base や AI の例題などに見られる Integrity Constraint(正リテラルのない節) に対し、General Logic Program の安定モデルは複数存在する可能性があるため、モデルごとに条件チェックする必要があるが、Magic Set 法によるインタプリタでは、この IC チェックを解探索と同時に行なうことにより、探索の効率化をはかることができた。

ボトムアップ法のみを使用する Magic Set 手続きは、ICOT で開発している並列プルーバ MGTP の問題に直接翻訳でき、並列化することができる。また、MGTP プルーバを高速化するために開発された様々な技術により、より効率のよい並列インタプリタが実現されることになる。

1 はじめに

Negation as Failure としての否定をもつ General Logic Program の宣言的意味として、安定モデルが 1988 年に提唱されてから [Gelfond, Lifschitz 88]、安定モデルの計算方法についての研究が、一部で注目されている。General Logic Program の安定モデルの存在問題は決定不能であるため、一般に安定モデルを求める手続きは存在しない。しかし、最近応用対象として着目されている演繹データベースや AI の問題の多くは有限領域であり、安定モデルの存在問題も決定可能であることから、有限領域のモデルを探す効率のよいアルゴリズムを発見する研究が行なわれてきている。本論文では、トップダウンとボトムアップの推論を演繹データベースの Magic Set 法を利用して統合することにより、有限領域の質問応答問題における全解探索が効率よく実行できる手続きを紹介する。この方法を使うと、General Logic Program 上の Abduction 問題も効率的に解くことができることもわかっている。また、インタプリタが 1 ページ以下の Prolog プログラムにおさまるという特徴もある。

この論文の以後の構成は、以下のようになる。第 2 章で、本論文で使用する記号および、問題領域を定義する。第 3 章で Prolog を質問応答の解探索プロシージャとする安定モデル Prolog のインタプリタの構成方法について述べ、第 4 章でこのインタプリタを拡張し、Magic Set 法を適用する方法について述べる。第 5 章では、Magic Set 法の能力について述べる。第 6 章では Abduction への応用について述べる。第 7 章では ICOT で開発している並列プルーバ MGTP を使用したインタプリタの並列化について述べる。第 8 章で結論を述べる。

```
vanilla(G,M):- vi(G,[],M).
vi(true,Mi,Mi):- !.
vi((G,GL),Mi,Mo):- !, vi(G,Mi,M1),vi(GL,M1,Mo).
vi((G1;G2),Mi,Mo):- !, vi(G1,Mi,Mo);vi(G2,Mi,Mo).
vi(~G,Mi,Mo):- !,(vi(G,[],_) -> fail;Mo=[~G|Mi]).
vi(G,Mi,[G|Mo]):- clause(G,SG),vi(SG,Mi,Mo).
```

図 1: Q/A に利用されたモデルを返す Prolog のバニラインタプリタ

2 準備

3 Prolog による安定モデル Prolog インタプリタ

安定モデル上での Prolog プログラムにおける Q/A 手続きは、以下の 2 つの条件を満たすモデルを探すことに相当する。

(a) Query に対する解が含まれること

(b) 安定モデルであること

(a) の条件に対しては、すべての解を求めることは重要であるが、(a) の条件さえ満たせば、(b) の問題は存在問題であり、複数あってもその 1 つが発見できればよい。ここでは、この 2 つのタスクを生成（解の発見）−テスト（安定モデル存在）の一連の手続きにシリアライズすることで単純化を行なう。この方法は、Query の Answer を含まない無駄なモデルの生成をいっさい行なわないためデータベースや AI などの応用では、大幅な効率化につながる。

解の発見

安定モデルに含まれるかどうかを別にすれば、解の発見は、通常の Prolog の実行で行なうことができる。ただし、解を導く証明木にあらわれるすべての正、負リテラルが安定モデルに含まれることが必要であるため（安定モデル生成のボトムアップ手続きにとって初期モデルになる）、解とともに次の安定モデルの存在チェック手続きに渡される。この初期モデルを返すインタプリタは、Prolog 上では、図 1のように簡単である。また、プログラム自身を変換してモデルを返すようにすれば、インタプリタを使用しなくてもよい。

解を含む安定モデルの存在

安定モデルの存在の証明は、Least Fixpoint を求めるためのボトムアップ手続きによって構成的に行なわれる。General Logic Program の安定モデルをボトムアップ法により求める手続きは、井上 [Inoue 92] らが発表しているが、我々もこの方法をほぼ利用する。ここで、ほぼという表現を利用したのは、仮説推論の K オペレータを陽に利用しないからである。[Inoue 92] で使用されている K オペレータを使用しないことにより、必要な *Axiom* がより単純になる[1]という小さな効果がある。我々の方法による安定モデルの求め方では、否定を含むプログラムは、以下のように Disjunctive なプログラムに変換される。

$$P :- Q_1, \ldots, Q_i, \tilde{}Q_{i+1}, \ldots, \tilde{}Q_n. \Rightarrow$$

$$Q_1, \ldots, Q_i \rightarrow P, \tilde{}Q_{i+1}, \ldots, \tilde{}Q_n | \tilde{}\tilde{}Q_{i+1} | \ldots | \tilde{}\tilde{}Q_n \quad (1)$$

ここで、$\tilde{}$ は、negation as failure をあらわす。$\tilde{}\tilde{}P$ は、直観的には $\tilde{}P$ の証明がない、すなわち P が存在することを意味する。$\tilde{}$ に関係した Axiom は、(2), (3) として表される。ただし、$\mathcal{M}$ は、安定モデルをあらわすものとする。(3) は、Model$\mathcal{M}$ に P がなければ、$\tilde{}P$ が assumable になり、(2) が満たされなくなることから導かれる条件である。

$$\forall P \in \mathcal{M}(\tilde{}P \notin \mathcal{M}) \quad (2)$$

[1] 井上らの方法では、(2) のかわりに $A, \neg \mathbf{K}A \rightarrow$、$\mathbf{K}A, \neg \mathbf{K}A \rightarrow$ が必要である。

```
satisfiable :-
        is_violated(C),!,
        satisfy(C),
        satisfiable.
satisfiable.

is_violated(C) :-
        (A--->C),
        A, not C.

satisfy(C) :-
        component(X,C),
        casserta(X),
        on_backtracking(cretract(X)),
        not false.

component(X,(Y;Z)) :-
        !, (X=Y ; component(X,Z)).
component(X,X).

on_backtracking(_).
on_backtracking(X) :-
        X,!, fail.

casserta((A,B)):- !, asserta(A),casserta(B).
casserta(A):- asserta(A).

cretract((A,B)):-
        (retract(A),fail;cretract(B)).
cretract(A):- retract(A).
```

図 2: SATCHMO のインタプリタ

$$\forall^{\neg} P \in \mathcal{M}(P \in \mathcal{M}) \tag{3}$$

(2) は、そのままボトムアップ手続きのルールとして表現可能である。(3) については、生成されたモデルに対する Integrity Constraint(:- ¯¯X, not X.) になる。

最も手軽でエレガントな基底モデルを求める Prolog プログラムの一つが SATCHMO [Bry 88] である。SATCHMO はもともとデータベースの Integrity Constraint チェッカの手続きとして開発されたわずか数行の Prolog プログラム (図 2) であり、range restricted[2] な一階述語論理問題を解くことができる。このプログラムを利用すると、安定モデル探索の手続きのうち、(3) の条件を除く部分がカバーされる。条件 (3) については、得られたモデルに対し、上の Integrity Constraint のチェックを行なう。

インタプリタの述語は、Query を G として、

```
glp(G):- vanilla(G,M), assert_models(M), satisfiable, exist_evidence.
```

となる。assert_models(_) が、バックトラック時に assertion をすべて消すようにすれば、Prolog の alternative として解をすべて得ることができる。

4 Magic Set 法によるインタプリタ

4.1 Magic Set 法の応用

前章のインタプリタでは、(3) 解の発見の部分を Prolog がトップダウンに実行しているが、幅優先の制御を入れない限り Prolog の探索は不完全である。これに引き替え、ボトムアップ法は、完全性はあるものの、Q/A に関係のない無駄な推論を行なってしまう。トップダウンの情報を利用して、ボトムアップ法が無駄なモデル生成を行なわないようにする技術として、マジックセット法がある [Bry 90, Fuchi 88]。我々の方法では、前半の Prolog 実行の部分にこれが適用される。

Magic Set 法を利用すると、すべての推論がボトムアップアルゴリズムで行なわれることになり、トップダウンからボトムアップへのモデルの受渡しも必要でなくなる。

Magic Set 法による解の探索は以下の 2 つから構成される。最初に与えられる問題は、Fact F、ルール P の 2 つの集合と、ゴール G であることとする。

(1) P、G から別記する方法によりマジックセット MS を生成する

(2) ボトムアップ法により、$MS \cup F \cup \{\mathtt{G} \rightarrow \mathtt{goal(G)}\}$ から、goal(G) がモデルに現れるまでモデル生成を行なう

[2] 節のすべての変数が前提部に出現する。

```
naf(G):-
        on_backtracking(abolish(found,1)),
        query(G),
        satisfiable,% Satchmo's clause
        (not not_exist_evidence),
        asserta(found(G)).

query(G):- goal(G),not found(G).
query(G):-
        magic_rule_is_violated(C),!,
        satisfy(C), % SATCHMO's clause
        query(G).

magic_rule_is_violated(C) :-
        (A---->C),
        A, not C.

false:- (~(X)), X.

not_exist_evidence:- (~ ~X),not (call(X)),!.
```

図 3: Magic Set 用インタプリタ

ただし、(1) は実行前に問題をコンパイルする形で行なわれる。また、ゴールと関係のないプログラムの Magic 規則は作る必要はない。

以上の手続きにより、Q/A の部分が完了することになる。ここで注意するのは、最初のボトムアップ推論は F と MS のみを利用して行なわれることである。この部分は、図 3のように SATCHMO のコードがほとんど利用できる。このインタプリタも全解探索を行なう。

4.2 Magic Set の生成

否定を含む節のマジックセットは、推論効率をあげるための工夫を除けば、(4) のようにして作られる。これらの節は、推論を始める前にあらかじめ生成されることになる。ここで GP は、P の否定を意味する新述語[3]で、P の述語名のみを変更したものである。

$$
\begin{aligned}
P &\leftarrow Q_1, \ldots, Q_i, \neg Q_{i+1}, \ldots, \neg Q_n \\
&\Rightarrow \\
GP &\leftarrow GQ_1 \\
GP, Q_1 &\leftarrow GQ_2 \\
&\cdots \\
GP, Q_1, \ldots, Q_i &\leftarrow P, \neg Q_{i+1}, \ldots, \neg Q_n | \neg Q_{i+1} | \ldots | \neg Q_n
\end{aligned}
\tag{4}
$$

4.3 Integrity Constraint の導入

Data Base や AI の例題などに見られる Integrity Constraint(正リテラルのない節、以後 IC) 条件は、Negation as Failure がなければ、節集合全体の性質として Global に考えることができる。しかし、General Logic Program の安定モデルは複数存在する可能性があるため、モデルごとに IC 条件はチェックする必要がある。Magic Set 法によるインタプリタは、この IC チェックを解の探索と同時に行なうことができる。IC 節を (5) のように Magic Set 化してしまうのである。この Magic Set ルールは、Q/A の手続きと並行して行なうことによって副次的に枝刈りの効果が期待できる。安定モデル探索時にも、この Magic Set はそのまま使用される。

$$
\begin{aligned}
P_1, \ldots, P_i, \neg P_{i+1}, \ldots, \neg P_n &\leftarrow \\
&\Rightarrow \\
&\leftarrow GP_1
\end{aligned}
$$

[3]Magic Set では magic- を頭につけて表すことが多い。

$$P_1 \quad \rightarrow \quad GP_2 \tag{5}$$
$$\cdots$$
$$P_1, \ldots, P_i \quad \rightarrow \quad false|\neg P_{i+1}|\ldots|\neg P_n$$

5 Magic Set 法の能力

これまでに述べた方法は、安定モデル上の解の探索について、以下のようなすぐれた性質を持つ。

Theorem 1 *general logic program* の質問応答問題の *Magic Set* 法による探索は、公平なボトムアップ手続きにより、有限の安定モデル上に解があれば、それをすべて求めることができる。

ここで紹介したSATCHMOのインタプリタは公平ではないが、原論文ではレベルをつけることによって公平にしたインタプリタもある。

6 並列化

ICOTでは、SATCHMOをベースに、range restrictedな問題を対象とした並列定理証明器MGTP[HFujita91]をKL1上に開発している。ここで述べたメタインタプリタは、MGTPをほとんんどそのまま使用することにより、並列に動作させることができる。MGTPは、range restricted条件をうまく活用することにより、disjunctive clauseによって起こる場合分けの部分を並列化している。否定を含む節はNon-Horn節に変換されるため、これにより起こる場合分けは、すべてこの並列化の恩恵を受けることができる。

Magic Setへの翻訳では、magic規則がrange restrictedでない場合が起こる。一方、KL1の言語的な制約により、問題の記述はrange restrictedでなければMGTPを使うことができない。この問題に対しては、データベースの研究で生まれたadornment法を使用すると、一部回避することができる。

7 結論

General Logic Programの有限安定モデル上の質問応答システムを、ボトムアッププルーバSATCHMOをベースに開発した。インタプリタは1ページ以下のPrologプログラムであり、Abductionやデータベースなど応用範囲も広い。また、並列プルーバMGTPを使用することにより、並列化が容易に行なえる。

この論文では議論しなかった重要な問題として、以下のような点について現在検討を行なっている。

(1) 包摂(subsumption)テストによるモデルの数の削減
[Stickel 92]では、Magic Set法における包摂テストについて精密に議論している。General Logic ProgramのMagic Set法においても、問題によっては包摂テストは非常に有効であり、取り込む必要がある。

(2) Non-Horn Clauseを使用した場合分けの順序
Non-Horn Clauseに翻訳された否定を含む節を使った場合分けの順序は、冗長な探索を防ぐ上で大きな意味を持つ。MGTPで我々が開発しているDisjunctive Modelの部分反駁の技術[MFujita92b]がこの問題に使用でき、その効果について検討を始めたところである。

(3) Non-Horn Logic Programmingとの間の技術関係
我々の安定モデルの計算方法は、Non-Horn Logic Programmingの計算方法と非常によく似ている。Non-Horn Logic Programmingとの技術的、論理的関係を調べることにより、互いの工夫が応用できる可能性もあり、重要な検討項目である。

謝辞

ICOTの井上、佐藤健両氏には、General Logic Programについての様々な質問に忍耐づよく答えていただいたことに感謝します。この研究を励まし、議論して下さった古川次長に感謝します。最後に、研究の機会を与えて下さり、深い知識と直観で我々を導いて下さった淵所長に感謝します。

参考文献

[Bry 88] Bry, F., "SATCHMO:a theorem prover implemented in Prolog," in Proc. CADE-9, pp415-434.

[Bry 90] Bry, F., "Upside-down Deduction," in Proc. 6th PODS, 1990.

[Fuchi 88] Fuchi, K., "Logical Source of Magic Set," rejume for ICOT meeting(in Japanese), 1988.

[Gelfond, Lifschitz 88] , Gelnfond, M., Lifschitz, V., "The Stable Model Semantics for Logic Programming," Proc. of ICLP'88, pp1070-1080, 1988.

[HFujita91] Fujita, H., and Hasegawa, R., "A Model Generation Theorem Prover in KL1 Using Ramified-Stack Algorithm," in *Proc. of ICLP91*, pp535-562.

[Inoue 92] Inoue, K., Koshimura, M., Hasegawa, R., "Embedding negation as failure into a model generation theorem prover," to appear in *Proc. of CADE-11*,1992.

[MFujita92a] Fujita, M., "Non-Horn Magic Set and Its Applications," to appear, 1992.

[MFujita92b] Fujita, M., "Dynamic Optimization of Choosing Disjunctive Models in Model Generation Theorem Prover1," to appear, 1992.

[Stickel 92] Stickel, M, "Upside-Donw Meta-Interpretation of the Model Elimination Theorem-Proving Procedure for Deduction and Abduction," to appear in *Journal of Automated Reasoning*, 1992.

付録 1 NAF-Prolog インタプリタ

```
%%%%% Stable Model Prolog Interpreter %%%%%%%%%%%
naf(G):- vanilla(G,M),exist_stable_model(M).
% prolog meta interpreter with memoization of lenmmas
vanilla(G,M):- vi(G,[],M).
vi(true,Mi,Mi):- !.
vi((G,GL),Mi,Mo):- !, vi(G,Mi,M1),vi(GL,M1,Mo).
vi((G1;G2),Mi,Mo):- !, vi(G1,Mi,Mo);vi(G2,Mi,Mo).
vi(~G,Mi,Mo):- !,(vi(G,[],_) -> fail;Mo=[~G|Mi]).
vi(G,Mi,[G|Mo]):- clause(G,SG),vi(SG,Mi,Mo).
% axiom
false:- call(~X),call(X).
% integrity checker based on satchmo
exist_stable_model(M):-
          assert_naf(M),
          satisfiable, % Satchmo's clause
          \+not_exist_evidence, !.
% evidence check
not_exist_evidence:- call(~~X),\+(call(X)),!.
% assert proof tree
assert_naf([G|GL]):- !,
        (call(G) -> true;assert(G)),% with subsumption test
        on_backtracking(retract(G)),
        assert_naf(GL).
assert_naf([]).
```

付録 2 Magic Set NAF-Prolog インタプリタ

```
%%%%% Stable Model Prolog Interpreter by Magic Set %%%%%
naf(G):-
        on_backtracking(abolish(found,1)),
        query(G),
        satisfiable,% Satchmo's clause
        (not not_exist_evidence),
        asserta(found(G)).
% revised SATCHMO
query(G):- goal(G),not found(G).
query(G):-
        magic_rule_is_violated(C),!,
        satisfy(C), % SATCHMO's clause
        query(G).
magic_rule_is_violated(C) :-
        (A---->C),
        A, not C.
% axiom
false:- call(~X),call(X).
% evidence check
not_exist_evidence:- call(~~X), not call(X),!.
```

付録 3 例題

(1) 有名な例題

四つ巴の有名な例題。トップダウンには解くことができない。

```
%%%%% Result %%%%%
?- naf(p(X)).
X=1;

X=3;

X=2;

X=4;

no

%%%%%% A Famous Problem %%%%%%
%p(1):- ~p(2).
%p(2):- ~p(3).
%p(3):- ~p(4).
%p(4):- ~p(1).
%%%%% Bottom Up Rule %%%%%
true ---> ((p(1), (~p(2)));(~ ~p(2))).
true ---> ((p(2), (~p(3)));(~ ~p(3))).
true ---> ((p(3), (~p(4)));(~ ~p(4))).
true ---> ((p(4), (~p(1)));(~ ~p(1))).

%%%%% Magic Set %%%%%
p(X) ----> goal(p(X)).
true ----> gp(X).
gp(1) ----> ((p(1), (~p(2)));(~ ~p(2))).
gp(2) ----> ((p(2), (~p(3)));(~ ~p(3))).
gp(3) ----> ((p(3), (~p(4)));(~ ~p(4))).
gp(4) ----> ((p(4), (~p(1)));(~ ~p(1))).
```

(2) 予約問題

これは、井上氏のオリジナルな問題である。Prologをトップダウン実行の部分に使用したインタプリタは、約0.85秒程で終了した。環境は、sun3、Sicstus Prologでコンサルトしたものである。

```
%%%%% Result %%%%%
% query
?- naf(holdmeeting(V1,V2,V3,V4)).
V1=mon,
V2=213,
V3=self,
V4=self ?;

no

%%%%% Sample of Room Reservation %%%%%%
% facts
person(mark).
```

```
person(donald).
day(mon).
day(tue).
room(212).
room(213).
reserved(212,mon).
reserved(213,tue).
busy(mark,agent,mon).
busy(donald,self,tue).

% rules
cand(X,self,Y) :-
        person(X),
        day(Y),
        ~busy(X,self,Y).
person(X), day(Y) ---> cand(X,self,Y),~(busy(X,self,Y));
                       ~(~(busy(X,self,Y))).
cand(X,agent,Y) :-
        busy(X,self,Y),
        ~busy(X,agent,Y).
busy(X,self,Y) ---> ~(busy(X,agent,Y)),cand(X,agent,Y);
                    ~(~(busy(X,agent,Y))).

openroom(Room,Day) :-
        day(Day),
        room(Room),
        ~reserved(Room,Day).
day(Day), room(Room) ---> ~(reserved(Room,Day)),openroom(Room,Day);
                          ~(~(reserved(Room,Day))).
selves_meeting :-
        holdmeeting(Day,Room,self,self).
holdmeeting(Day,Room,self,self) ---> selves_meeting).
holdmeeting(Day,Room,self,self) :-
        cand(mark,self,Day),
        cand(donald,self,Day),
        openroom(Room,Day).
cand(a,self,Day), cand(b,self,Day), openroom(Room,Day)
               ---> holdmeeting(Day,Room,self,self).
holdmeeting(Day,Room,Ida,Idb) :-
        cand(mark,Ida,Day),
        cand(donald,Idb,Day),
        openroom(Room,Day),
        ~selves_meeting.
cand(a,Ida,Day), cand(b,Idb,Day), openroom(Room,Day)
               ---> ~(selves_meeting),holdmeeting(Day,Room,Ida,Idb);
                    ~(~(selves_meeting))).
```