知能と複雑系 119−19
(2000. 1. 13)

SAT ソルバによるアクション言語処理系の実装

鍋島 英知[1] 井上 克巳[1,2] 羽根田 博正[1,2]

[1]神戸大学大学院自然科学研究科
[2]神戸大学工学部電気電子工学科

[1]〒657-8501 神戸市灘区六甲台町 1-1
神戸大学大学院 自然科学研究科 情報メディア科学専攻
TEL: (078)881-1212 内線 5243 FAX: (078)881-3193
e-mail: nabesima@s2.eedept.kobe-u.ac.jp

あらまし

最近，高速なプランニングアルゴリズムの1つとして，プランニンググラフと呼ばれるデータ構造を用いた手法が研究されている．それは，プラン探索空間をいったんプランニンググラフに符号化した後，充足可能性問題 (SAT) に変換し，高速な SAT ソルバにより解くという手法である．本研究では，このプランニンググラフと SAT ソルバによるプラン抽出法を基に，アクション言語処理系 AMP を実装した．AMP では，アクション言語 $\mathcal{A}$ による領域記述に対し，プランニングの他にも，予測や推定，モデル生成（初期状態の推定）等を問い合わせることができる．

キーワード アクション言語，プランニング，SAT，プランニンググラフ

Implementing an Action Language using a SAT Solver

Hidetomo Nabeshima* Katsumi Inoue*,** Hiromasa Haneda*,**

*Graduate School of Science and Technology, Kobe University
**Department of Electrical and Electronics Engineering,
Faculty of Engineering, Kobe University

*Division of Information and Media Science,
Graduate School of Science and Technology, Kobe University
Rokkodai, Nada Kobe 657-8501 Japan
TEL: +81 78 881 1212 (5243) FAX: +81 78 881 3193
e-mail: nabesima@s2.eedept.kobe-u.ac.jp

Abstract

In recent years the research of planning algorithm has advanced drastically. First of all, the approach encodes plan search space into the data structure called planning graph. Next the graph is transformed into the satisfiability problem (SAT), and then is solved by a high-performance SAT solver.

In this study, we implemented an action language processing system AMP on the basis of planning graph and the plan extraction by a SAT solver. Using this system, it becomes possible to do planning and model generation (estimation of the initial state) for a domain description written in the action language $\mathcal{A}$.

key words Action languages, planning, SAT, planning-graph

1 はじめに

ここ数年プランニングアルゴリズムの研究は，大きな進展をみせている．その発端となったのが Blum と Furst による Graphplan[1] である．

Graphplan は，STRIPS 形式のプランニング問題を入力とし，目標条件を達成する半順序プランを出力するプラン生成器である．Graphplan の特徴は，プラン探索空間を，いったん**プランニンググラフ**と呼ばれるデータ構造に符号化することにある．プランニンググラフでは，互いに相容れないアクション間・フルーエント間の関係（**相互排他関係**，mutex relation）がチェックしてある．次にプランニンググラフからプランを体系的探索アルゴリズムにより抽出する．このとき，先に調べておいた相互排他関係が探索空間を大きく絞り込むため，非常に効率よくプランを発見することできる．

しかし大規模な問題においては，相互排他関係を活用しても体系的探索アルゴリズムによるプラン抽出に時間がかかる．そこで Kautz と Selman は，プランニンググラフが**充足可能性問題** (Satisfiability; **SAT**) に自動変換できることを示し [5]，既存の高速な SAT ソルバによりプランを抽出するプラン生成器 Blackbox を提案した [6]．現在のところ，Blackbox は最速のプラン生成器の 1 つである．

本研究の目的は，プランニンググラフと SAT ソルバによる高速なプラン生成法を基に，**アクション言語処理系**を実装することにある．

アクション言語とは，自然言語表現を用いてアクションによる状態の変化を記述する形式的モデルである．アクション言語は状態変化領域を記述するための言語であるが，その記述に対してプランやモデルを問い合わせるための言語でもある．

本論文では，プランニンググラフと SAT ソルバによる手法が，プランニングだけでなく，モデル生成（初期状態の推定）や，ある状態においてあるフルーエントが成立するかどうか（予測），ある状態においてどのようなフルーエントが成立するかどうか（推定）などの問い合わせに対しても有効であることを示す．

これまで，アクション言語処理系実装のための手法として，アクション言語による記述を拡張論理プログラム等の別の枠組みに変換する手法が提案されているが，アクション言語による記述を直接解釈し，効率良く処理するための手法は提案されていない．

本論文で紹介するアクション言語処理系 AMP は，入力として言語 $\mathcal{A}$ [3] による記述を受け取り，プランニンググラフと SAT ソルバにより，各種問い合わせに応答する．AMP の実装には Java 言語を使用した．他の高速なプラン生成器の多くは C 言語で実装されているため，速度面で Blackbox には少し劣るが，拡張性・再利用性，プロトタイプ作成の容易さやといった点では優れていると考える．

本論文の残りは次のようになっている．まず，アクション言語 $\mathcal{A}$ について続く 2 章で紹介し，3 章で AMP の実装アルゴリズムを示す．4 章で実験結果について考察する．

2 アクション言語 $\mathcal{A}$

アクション言語 $\mathcal{A}$[1] を 4 つ組 $\langle \mathbb{A}, \mathbb{F}, \mathbb{E}, \mathbb{V} \rangle$ で表す．ここで $\mathbb{A}$ は**アクション名**の集合，$\mathbb{F}$ はフルーエント名の集合，$\mathbb{E}$ は**効果命題** (effect proposition) の集合，$\mathbb{V}$ は**評価命題** (value proposition) の集合である．

言語 $\mathcal{A}$ において，フルーエントは真または偽の 2 値をとる．フルーエント名 F の否定 $\neg F$ を**負のフルーエント**といい，F を**正のフルーエント**という．フルーエントを小文字の f で表し，フルーエント名を大文字の F で表す．

効果命題は，アクションとその効果の因果関係を記述する：

$$a \textbf{ causes } \phi \textbf{ if } \psi \tag{1}$$

ここで a はアクション名，ϕ はフルーエントの連言，ψ はフルーエントを論理演算子 $(\vee, \wedge, \neg, \supset, \equiv)$ で結合した式である．ϕ をアクションの**効果**と呼び，ψ をアクションの**前提条件**と呼ぶ．

評価命題は，観測した事実を記述する：

$$\phi \textbf{ after } a_1; \ldots; a_m \quad (m \geq 0) \tag{2}$$

ここで ϕ はフルーエントの連言，a_i はアクション名である．

いくつか有用な省略形を定義する．ここで *True* は，恒真を意味する特別なフルーエントで，全ての状態で真となる．*False* $= \neg$*True* である．効果命題 (1) に対し，前提条件 ψ が *True* ならば，**if** 以下を省略する：

$$a \textbf{ causes } \phi$$

評価命題 (2) に対し，$m = 0$ であるならば，簡単に次のように記述する：

$$\textbf{initially } \phi \tag{3}$$

言語 $\mathcal{A}$ による**領域記述**は，効果命題と評価命題の集合である．**状態** σ を，全てのフルーエント名につい

[1] 本論分で定義する言語 $\mathcal{A}$ は，文献 [3] による定義に若干拡張を加えたものであるが，表現能力においては等価である．

て，正または負のフルーエントのいずれかを含む集合と定義する：

$$\sigma = S \cup \{\neg F \mid F \in \mathbb{F} \setminus S\}.$$

ここで $S \subseteq \mathbb{F}$ である．任意の状態 σ と，フルーエント f のみからなる式 $\phi = f$ に対し，$f \in \sigma$ であるとき，かつそのときに限り，**状態 σ で式 ϕ が真**であると定義する．任意の式については，論理演算子の真偽値表に従って拡張する．

解釈 I は，σ_0 を初期状態，Φ を遷移関数とすると，その対 (Φ, σ_0) である．**遷移関数** Φ は，アクション名 a と状態 σ の対 (a, σ) を状態へ写像する．任意の解釈 I と，任意のアクション列 $a_1; \ldots; a_m$ に対して，$I^{a_1;\ldots;a_m}$ は次の状態を与える：

$$I^{a_1;\ldots;a_m} = (\Phi, \sigma_0)^{a_1;\ldots;a_m} = \Phi(a_m, \Phi(a_{m-1}, \ldots, \Phi(a_1, \sigma_0)\ldots)).$$

評価命題 (2) に対し，状態 $I^{a_1;\ldots;a_m}$ で ϕ が真であれば，評価命題 (2) が解釈 I において**真**であるという．

解釈 I が領域記述 D の**モデル**であるのは，(i) D に含まれるすべての評価命題が I において真であり，(ii) すべてのアクション名 a とすべての状態 σ に対し，遷移関数が以下の条件を満たす場合である：

(a) もし D が，状態 σ で前提条件を満たす効果命題 (1) を含むならば，かつそのときに限り，$\phi \in \Phi(a, \sigma)$ である．

(b) もし D が，そのような効果命題を含まなければ，$f \in \sigma \Leftrightarrow f \in \Phi(a, \sigma)$ である．

条件 (b) は**慣性の法則**を表している．上の条件を満たす遷移関数は，多くとも1つしか存在しない．よって，同じ領域記述における異なったモデルは，そのモデルの初期状態によってのみ異なる．ある評価命題が領域記述 D のモデル M で真であるならば，

$$M \models (\phi \textbf{ after } a_1; \ldots; a_m).$$

と記述する．任意のモデルにおいて真であるならば，

$$D \models (\phi \textbf{ after } a_1; \ldots; a_m).$$

と記述する．

3 AMP

アクション言語 $\mathcal{A}$ による領域記述 D に対し，AMP では次の4種類の問い合わせが可能である：

$$D \models \phi \textbf{ after } a_1; \ldots; a_m \quad (4)$$

$$D \models X \textbf{ after } a_1; \ldots; a_m \quad (5)$$

$$D \models \psi \textbf{ after } X \quad (6)$$

$$D \models \textbf{initially } X \quad (7)$$

式 (4) は，アクション列 $a_1; \ldots; a_n$ を実行後の状態において式 ϕ が成立するかの問い合わせである（**推定**）．式 (5) は，アクション列 $a_1; \ldots; a_n$ を実行後，どのようなフルーエントが成立するかという問い合わせである（**予測**）．式 (6) は，目標条件 ϕ を成立させるアクション列 X の問い合わせとなる（**プランニング**）．式 (7) は，初期状態で成立するフルーエントの問い合わせである．これは，領域記述 D のモデル $M = (\Phi, \sigma_0)$ の初期状態 σ_0 を求めることに等しい．遷移関数 Φ は効果命題から簡単に求めることができるので，AMP では式 (7) の問い合わせを**モデル生成**と呼ぶ．

以下では，AMP の重要な機能であるプランニングとモデル生成について紹介する．

3.1 プランニング

言語 $\mathcal{A}$ による領域記述を D とする．ここでは簡単のために，D の初期状態で成立するフルーエントは，式 (3) の形式の評価命題により完全に記述されているものと仮定する[2]．初期状態 σ_0 は次式で与えられる：

$$\sigma_0 = \bigcup_{(\textbf{initially } f_1 \wedge \ldots \wedge f_n) \in D} \{f_1, \ldots, f_n\}$$

また，前提条件に選言を含む効果命題が領域記述にある場合，その前提条件を以下のように選言標準形に変換し，

$$a \textbf{ causes } \phi \textbf{ if } \psi_1 \vee \ldots \vee \psi_n$$

これを以下の効果命題と置き換える：

$$\begin{array}{l} a \textbf{ causes } \phi \textbf{ if } \psi_1 \\ \quad \vdots \\ a \textbf{ causes } \phi \textbf{ if } \psi_n \end{array} \quad (8)$$

次に，あるアクション名に関する効果命題が唯1つになるように，アクション名を変更する．例えば，アク

[2] D が式 (2) の形式の評価命題を含んでいる場合，初期状態を求めるためにはモデルを生成する必要がある．モデル生成については 3.5 節を参照．

ションa に関する効果命題が，式 (8) のように n 個あったとすると，それを異なるアクション名 $a_1, \ldots, a_n$ に関する効果命題に変換する：

$$a_1 \textbf{ causes } \phi \textbf{ if } \psi_1$$
$$\vdots$$
$$a_n \textbf{ causes } \phi \textbf{ if } \psi_n$$

これにより以下では，アクション名と効果命題が一対一対応するので，効果命題のことを“アクション”と呼ぶ．

領域記述と初期状態と目標条件が与えられると，AMP は，プランが見つかるまで次の 2 つのステップを繰り返しながら動作する：

Step 1. プランニンググラフの展開

Step 2. プランニンググラフからプラン抽出

3.2 プランニンググラフ展開

プランニンググラフとは，レベル付きグラフである[3]．偶数レベルの頂点はフルーエントに対応し，奇数レベルの頂点はアクションに対応する (図 1)．AMP におけるプランニンググラフ展開アルゴリズムは，Graphplan のそれに従う．前節で示したように，前提条件に含まれる選言を展開し，アクション名と効果命題に一対一対応を与えることで，プランニンググラフ展開アルゴリズムに修正を加えることなく，前提条件に選言を含むアクションを扱えるようになる．

プランニンググラフは以下のようにして作られる．ここでレベル i の頂点集合を L_i で表す．

L_0 は，初期状態において真であるフルーエントからなる．奇数レベル i の頂点集合 L_i は，次のアクションからなる：

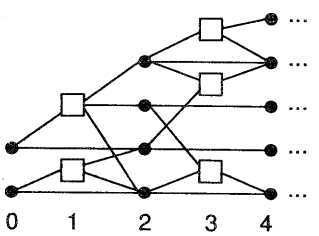

図 1: プランニンググラフ

- L_{i-1} で前提条件を満たすすべてのアクション
- L_{i-1} に含まれるフルーエントを L_{i+1} へ伝播する no-op アクション[4] (各 $f \in L_{i-1}$ に対し，アクション *no-op* **causes** f **if** f を追加する)．

アクションとその前提条件の間は辺で結ばれる．L_{i+1} は，L_i に含まれるすべてのアクションの効果からなる．アクションとその効果の間は辺で結ばれる．

同じレベルにある任意の 2 頂点について，以下で定義される**相互排他関係** (mutex relation) をチェックする (図 2)：

- **レベル i の 2 つのアクションが相互排他関係にあるのは，**
  1. 片方の効果が他方の効果を否定する場合 (inconsistent effects)，または，
  2. 片方の効果が他方の前提条件を否定する場合 (interference)，または，
  3. 2 つのアクションの前提条件が，レベル $i-1$ で相互排他関係にある場合 (competing needs).
- **レベル i の 2 つのフルーエントが相互排他関係にあるのは，**
  1. 片方が他方の否定である場合，または，
  2. 片方を導いたレベル $i-1$ のすべてのアクションが，他方を導いたレベル $i-1$ のすべてのアクションと相互排他関係にある場合 (inconsistent support).

あるレベル i における任意の 2 つのアクション (フルーエント) が相互排他関係にあるということは，レベル i においてその 2 つのアクション (フルーエント) が同時に成立しないことを意味する．このことが，プラン抽出ステップで有効に活用される．

3.3 プラン抽出

プランニンググラフの最大レベル l の頂点集合 L_l に，目標条件のすべてのフルーエントが含まれ，かつ，それらに相互排他関係がない場合，AMP はプラン抽出アルゴリズムを適用する．AMP のプラン抽出アルゴリズムは，Blackbox のそれに等しい．プランニンググラフは，以下の規則に従って SAT 問題に変換される．

[3] グラフがレベル付きであるとは，グラフの頂点を，互いに素な集合 $L_1, L_2, \ldots, L_n$ に分割できるときをいう．ここで L_i は，隣り合うレベル (L_{i-1}, L_{i+1}) の頂点につながっている頂点の集合である．

[4] 図中で no-op アクションは，偶数レベル同士を結ぶ水平線で表されている．

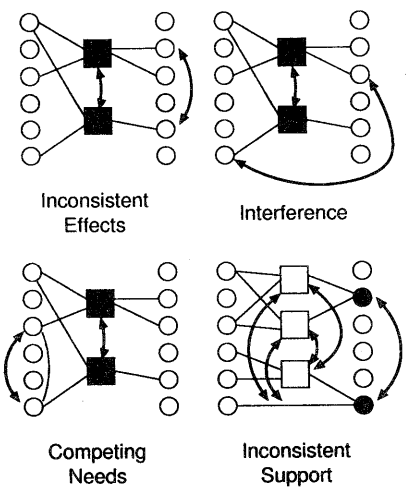

図 2: 相互排他関係 [8]

(1) 初期レベル $L_0 = \{f_1, \ldots, f_n\}$ はレベル 0 において真であり，目標条件 $\{g_1, \ldots, g_m\}$ はレベル l において真である：

$$f_1^0 \wedge \ldots \wedge f_n^0 \wedge g_1^l \wedge \ldots \wedge g_m^l$$

(2) レベル i において相互排他関係にあるアクション a, b は同時に成立しない：

$$\neg a^i \vee \neg b^i$$

(3) アクションは，その前提条件を含意する．すなわち，レベル i のアクション a と辺で結ばれているレベル $i-1$ のフルーエント $p_1, \ldots, p_n$ に対し，以下の式を追加する：

$$a^i \supset p_1^{i-1} \wedge \ldots \wedge p_n^{i-1}$$

(4) 偶数レベル i に含まれるフルーエントは，それらを効果として持つレベル $i-1$ のすべてのアクションの選言を含意する．すなわち，レベル i のフルーエント f と辺で結ばれているレベル $i-1$ のアクション $a_1, \ldots, a_n$ に対し，以下の式を追加する：

$$f^i \supset a_1^{i-1} \vee \ldots \vee a_n^{i-1}$$

例として，次の Yale Shooting 領域 [3] D に対し，

load **causes** *loaded*.

shoot **causes** $\neg$*loaded* $\wedge \neg$*alive* **if** *loaded*.

initially *alive* $\wedge \neg$*loaded*.

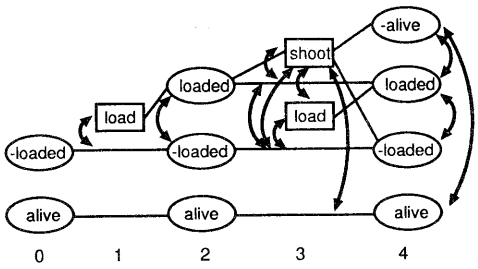

図 3: Yale Shooting 領域のプランニンググラフ

次式を満たすアクション列 X を求めてみる．

$$D \models \neg alive \textbf{ after } X$$

この領域記述に対し，レベル 4 までプランニンググラフを展開すると図 3 のようになる．さらに SAT 問題に変換すると次式になる：

$$\begin{aligned}
&(\neg loaded^0 \wedge alive \wedge \neg alive^4) \\
&\wedge (\neg load^1 \vee \neg nop^1_{loaded}) \\
&\wedge (\neg shoot^3 \vee \neg nop^3_{loaded}) \wedge (\neg shoot^3 \vee \neg nop^3_{\neg loaded}) \\
&\wedge (\neg shoot^3 \vee \neg nop^3_{alive}) \wedge (\neg load^3 \vee \neg nop^3_{\neg loaded}) \\
&\wedge (\neg shoot^3 \vee \neg load^3) \wedge (\neg nop^3_{loaded} \vee \neg nop^3_{\neg loaded}) \\
&\wedge (nop^1_{\neg loaded} \supset \neg loaded^0) \wedge (nop^1_{alive} \supset alive^0) \\
&\wedge (shoot^3 \supset loaded^2) \wedge (nop^3_{loaded} \supset loaded^2) \\
&\wedge (nop^3_{\neg loaded} \supset \neg loaded^2) \wedge (nop^3_{alive} \supset alive^2) \\
&\wedge (loaded^2 \supset load^1) \wedge (\neg loaded^2 \supset nop^1_{\neg loaded}) \\
&\wedge (alive^2 \supset nop^1_{alive}) \wedge (\neg alive^4 \supset shoot^3) \\
&\wedge (loaded^4 \supset load^3 \vee nop^3_{loaded}) \\
&\wedge (\neg loaded^4 \supset shoot^3 \vee nop^3_{\neg loaded}) \\
&\wedge (alive^4 \supset nop^3_{alive})
\end{aligned}$$

これを解くと，プラン $X = load^1; shoot^3$ を得る[5]．

3.4 初期状態の仮定

AMP では，初期状態で真偽値が不明なフルーエントに対し，**閉世界仮説** (CWA) を宣言することができる．CWA は，以下のようにしてを効率良く実装されている [8].

[5]正確には，奇数レベルで真となるアクションを取り出し，さらに冗長なアクションを削除した結果である．アクション a が冗長であるとは，プランからアクション a を取り除いても目標条件を達成できる場合をいう．

ある奇数レベル i において，追加したいアクションの前提条件に $\neg F$ が現れるのに，L_{i-1} に $F, \neg F$ が含まれない場合，レベル 0 からレベル $i-1$ までの各偶数レベルに $\neg F$ を追加し，各奇数レベルに $\neg F$ に関する no-op アクションを追加する[6]．

同様にして，初期状態で真偽値が不明なフルーエントを，真または偽と仮定してプランを生成することができる．これは次のように処理される．

ある奇数レベル i において，追加したいアクションの前提条件に f が含まれるのに，L_{i-1} に $f, \neg f$ が含まれない場合，レベル 0 からレベル $i-1$ までの各偶数レベルに $f, \neg f$ を追加し，各奇数レベルに f に関する no-op アクションと $\neg f$ に関する no-op アクションを追加する．ここで，各レベルに追加した 2 つのノードは相互排他関係にある．

この仮定の下で生成されたプランは，領域記述の特定のモデルにより帰結されるプランである．

3.5 モデル生成

AMP では，任意の領域記述のすべてのモデルを求めることができる．そのアルゴリズムは，プランニンググラフ展開アルゴリズムとプラン抽出アルゴリズムに制約を加えたものであるため，探索空間がプラン生成の場合よりも狭くなり，効率良くモデルを生成することができる．

言語 A による領域記述を D とする．D に含まれるすべての評価命題に対し，k 番目に実行されるすべてのアクションを取り出す関数 Act を以下で定義する：

$$Act(k) = \{a_k \mid (\phi \textbf{ after } a_1; \ldots; a_k; \ldots; a_n) \in D\}$$

次に，k 個のアクションを実行後，満たさなければならない条件を取り出す関数 $Goal$ を以下で定義する：

$$Goal(k) = \bigcup_{(f_1 \wedge \ldots \wedge f_n \textbf{ after } a_1; \ldots; a_k) \in D} \{f_1, \ldots, f_n\}$$

モデル生成のためのプランニンググラフ展開アルゴリズムは次のようになる．$L_0 = Goal(0)$ とする．奇数レベル $2i-1$ の頂点集合 L_{2i-1} は次のアクションからなる：

- $Act(i)$ に含まれるアクション
- L_{2i-2} に含まれるフルーエントを L_{2i} へ伝播する no-op アクション

もし，奇数レベル $2i-1$ に追加するアクションの前提条件に f が含まれるのに，L_{2i-2} に f が含まれない場合，レベル 0 からレベル $2i-2$ までの各偶数レベルに $f, \neg f$ を追加し，各奇数レベルに f と $\neg f$ に関する no-op アクションを追加する．ここで，各レベルに追加した 2 つのノードは相互排他関係にある[7]．$Act(i)$ に含まれるアクションは，レベル $2i-1$ において常に実行可能であるわけではないため，そのようなアクションであっても，グラフ作成のために前提条件を追加する必要がある．

また，奇数レベル i に追加するアクションの効果に f が含まれるのに，L_{i-1} に $f, \neg f$ が含まれない場合，レベル 0 からレベル $i-1$ までの各偶数レベルに $f, \neg f$ を追加し，各奇数レベルに f に関する no-op アクションと $\neg f$ に関する no-op アクションを追加する．ここで，各レベルに追加した 2 つのノードは相互排他関係にある．これは，レベル $i-1$ までの各偶数レベルにおいてフルーエント f の真偽値が不明であったことを言明するためである．

もし，$Act(i) = \emptyset$ であれば，グラフ展開を終了し，モデル抽出に移る．

AMP はモデル抽出のために，プランニンググラフを以下の規則に従って SAT 問題に変換する．規則 (2),(3),(4) はプラン抽出の規則と同じである．ここで，プランニンググラフの最大レベルを $2l$ とする．

(1') レベル 0 から $2l$ までの各偶数レベル $2i$ において，$Goal(i) = \{f_1, \ldots, f_n\}$ に含まれるフルーエントは真である：

$$f_1^{2i} \wedge \ldots \wedge f_n^{2i}$$

(5) 各奇数レベル $2i-1$ において真となるアクションは，$Act(i)$ に含まれるアクションか，または，no-op アクションである：$L_{2i-1} \setminus Act(i)$ から no-op アクションを除いた集合を $\{a_1, \ldots, a_n\}$ とすると，

$$\neg a_1^{2i-1} \wedge \ldots \wedge \neg a_n^{2i-1}$$

(6) 各奇数レベル $2i-1$ において，アクション $a \in L_{2i-1}$ が偽であるならば，その前提条件 ϕ は偽でなければならない：

$$\neg a^{2i-2} \supset \neg \phi^{2i-1}$$

[6] F は，レベル i においてアクションを追加するために初めて必要とされたフルーエント名であり，それまでのレベルの相互排他関係に影響を及ぼさない．

[7] もし，レベル 0 からレベル $i-1$ までの偶数レベルに $\neg f$ がすでに存在する場合，$\neg f$ を前提条件に含むアクションが存在するかもしれない．その場合，そのアクションとの相互排他関係もチェックする必要がある．

例えば評価命題 ϕ **after** $a_1;a_2;a_3$ に対し，規則 (1') は，アクション列 $a_1;a_2;a_3$ の実行後 ϕ が真となることを言明している．規則 (5),(6) は，奇数レベル 1,3,5 において，アクション a_1,a_2,a_3 が実行された，または，前提条件が満たされず実行できなかったことを言明している．

プランニンググラフは，初期状態で真偽値が不明なフルーエント f に対し，f または $\neg f$ をグラフに追加しながら作成されているので，この変換規則により得られる SAT 問題を解けば，すべての評価命題を真とする解釈の初期状態が求まる．すなわち，SAT 問題の解からレベル 0 で真となるフルーエントを取り出せばよい．

例として，因果関係が絡み合った領域記述を考える：

$push(b1)$ **causes** $on(s3)$ **if** $on(s1), on(s2)$.
$push(b2)$ **causes** $on(s1)$ **if** $on(s2), on(s3)$.
$push(b3)$ **causes** $on(s2)$ **if** $on(s3), on(s1)$.
$on(s3)$ **after** $push(b1); push(b2); push(b3)$.

この領域記述のモデルを求めると以下の 2 つのモデル（初期状態）を得る．ここで初期状態に含まれていないフルーエントは，真または偽どちらの値でも良いことを表している．

$\{on(s1), on(s2)\}$
$\{on(s3)\}$

この結果は，スイッチ $s1, s2$ がともに on であったか，もともとスイッチ $s3$ が on であったことを表している．このように AMP のモデル生成を利用することで，ある事象の観測結果から，その原因を推定することが可能になる．

4 考察

AMP の内部で使用される SAT ソルバは，命題論理の充足可能性問題が解くことのできるソルバであれば何でもよい．現在 AMP で使用している SAT ソルバは，最も基本的な体系的アルゴリズム DPLL[2] を実装したものである．これは AMP の動作検証用に試作したもので，その速度は決して速くない．Graphplan で用いられている体系的なプラン抽出アルゴリズムよりも遅い．今後，より高速な既存の SAT ソルバを AMP に組み込む予定である．

ここでは，AMP・Blackbox ともに，Graphplan で用いられている体系的なプラン抽出アルゴリズムを使用した場合の実験結果を示す[8]．

block-world.12	Time [sec]	Nodes
one model	2.19	1278
all models	14.61	1278
one plan	27.24	1842

表 1: モデル生成速度

Problem	Nodes	AMP [sec]	Blackbox [sec]
tire-world.01	533	1.43 (0.10)	0.27
tire-world.04	1036	3.74 (0.82)	0.55
block-world.12	1842	27.24 (19.34)	0.82

（実験環境 CPU:PentiumII 366MHz, Memory:128MB）

表 2: プラン生成速度

まず，AMP の特徴であるモデル生成の実験結果を示す（表 1）．Block-world.12 は，7 個のブロックを指定の形に積み上げる問題で，12 ステップのプランを要する．モデル生成では，プランと目標条件を与え初期状態を推定させた．表中の Nodes は AMP により作成されたプランニンググラフの頂点数である．

実験結果から，モデル生成で作成されるグラフのほうがプランニングで作成されるグラフよりも小さく，また処理時間も短いことが分かる．この問題では，すべてのモデルを求めた場合でも，1 つのプランを求める場合より速い結果となった．

次に，AMP と Blackbox のプランニング速度の比較実験結果を示す（表 2）．AMP は Java 言語で実装され，Blackbox は C 言語で実装されている．表中の Nodes は AMP により作成されたプランニンググラフの頂点数である．括弧内の数字は，プラン抽出にかかった時間を表している．

規模の小さな問題では 5 倍程度の速度差であるが，問題の規模が大きくなるにつれその差は開いている．特に，プランニンググラフのサイズが大きくなるにつれ，プラン抽出により多くの時間がかかっている．この点については，SAT ソルバによるプラン抽出法に置き換えることで，速度向上が期待できる．

実験結果からも分かるように，Java 言語による実装では，C 言語による実装と比べ速度面で不利である．しかし，より抽象度の高いプログラミングをすることが可能であること，メモリ管理に悩まされることがないこと等の理由から，複雑なアルゴリズムであっ

[8] AMP, Blackbox ともに，動作オプションを指定することにより，プラン抽出アルゴリズムを選択できる．とくに AMP では，モデル生成においても体系的な探索アルゴリズムが利用できる．

ても比較的容易に実装できる．この利点を生かし，今後，さらなる処理速度の向上や，より表現力豊かなアクション言語に対応させていく予定である．

プランニンググラフのサイズは，グラフ展開・プラン抽出アルゴリズムの双方に大きな影響を及ぼすため，グラフのサイズを抑えることは速度向上のための重要な要因である．AMP や Blackbox で用いられているプランニンググラフの展開アルゴリズムは，初期状態からグラフを順次展開していくが，特に目標条件を指向しているわけではない．なぜなら各偶数レベルにおいて実行可能なすべてのアクションを次の奇数レベルに加えているからである．そこで，初期状態から目標条件に向かうグラフ展開に加え，目標条件から初期状態へ向かうグラフ展開を行うことで，プランニンググラフのサイズを抑える研究がなされている [4]．我々もこれまでに，目標条件から初期状態へ向かうプラン生成システムを実装しているので [7]，今後これらを組み合わせてプランニンググラフのサイズの抑制を図る予定である．

謝辞 本研究の一部は文部省科学研究費補助金（特別研究員奨励費）による．

参考文献

[1] Blum, A. L. and Furst, M. L.: Fast planning through planning graph analysis, *Artificial Intelligence*, Vol. 90, No. 1–2, pp. 279–298 (1997).

[2] Davis, M., Logemann, G. and Loveland, D.: A Machine Program for Theorem Proving, *Communications of the ACM*, Vol. 5, No. 7, pp. 394–397 (1962).

[3] Gelfond, M. and Lifschitz, V.: Representing action and change by logic programs, *Journal of Logic Programming*, Vol. 17, No. 2–4, pp. 301–321 (1993).

[4] Kambhampati, S., Parker, E. and Lambrecht, E.: Understanding and extending Graphplan, *Proceeding of ECP-97: Recent Advances in AI Planning* (Steel, S. and Alami, R.(eds.)), LNAI, Vol. 1348, Berlin, Springer, pp. 260–272 (1997).

[5] Kautz, H. and Selman, B.: Pushing the Envelope: Planning, Propositional Logic and Stochastic Search, *Proceeding of AAAI-96*, pp. 1194–1201 (1996).

[6] Kautz, H. and Selman, B.: Unifying SAT-based and Graph-based Planning, *Proceedings of IJCAI-99*, pp. 318–325 (1999).

[7] 鍋島英知, 井上克巳, 羽根田博正: Java 言語によるアクション言語の処理系の実装, 第 13 回人工知能学会全国大会論文集, pp. 26–29 (1999).

[8] Weld, D. S.: Recent Advances in AI Planning, *AI Magazine*, pp. 93–123 (1999).