人工知能と対話技法 19－1
（1981. 1. 17）

# 階層的プロダクションシステムHIPSについて

大上勝也† 西山静男† 溝口理一郎†† 角所収††
（†関西大学工学部）（††大阪大学産業科学研究所）

## 1.はじめに

人工知能の分野において，知識の表現とその利用法が極めて重要な問題として注目を集めている中で，Production System (PS) [1]-[3]はDENDRAL[4]やMYCIN[5]など多くの実働しているシステムにも用いられ，有望な知識表現法と考えられている．

基本PSはstringに対する書き換えシステムとして考えられ，次の3つの基本的な構成要素から成っている．

(1) Working Memory(WM): 単なるstringの集まり．

(2) ProductionMemory(PM): $\alpha \rightarrow \beta$ というif-thenclauseの形式をとるルールの集合で，$\alpha$，$\beta$ 共にstringである．left-hand-side (LHS)即ち $\alpha$ を条件部，right-hand-side (RHS)即ち $\beta$ を実行部と呼び，WM内に $\alpha$ が現れたら $\beta$ に書き換えることを意味する．

(3) Production System Interpreter (PSI): PMのルールの中からWMに対して条件部がマッチするものを探し出し，WMを実行部の指示通りに書き換える働きをする．

PSの基本動作はあるWMに対して，その条件部がマッチするルールを探し，それを実行することの繰り返し(recognize-act cycleという)であるが，一般に起動可能なルールは複数個存在するので，実際にはこれらの内どのルールを起動するかを決定する必要がある．このようにある時点で同時に起動可能なルールの集合をconflict set，又この中からどれか1つのルールをある種の戦略に従って選択することをconflict resolutionと呼び，これらはすべてPSIが取り行なう．

基本的なPSには次に挙げる3つの特徴がある．

(1) Modularity: PM中の各々のルール間の相互作用はWMの変更(deposit, deleteなど)を通じてのみ間接的になされるので，他のルールとの関係を考慮せずに記述できる．そのためルールは自由に追加，除去，変更ができ，システムの能力を段階的に増大させていくことが可能である．

(2) Readability: 従来のプログラミング手法では使用される知識の内容が制御の流れの中に含まれているため，どのような知識がいかに使われたかを知るためには，そのプログラムの処理の流れを順次見ていかなければならない．ところがPM中の各々のルールはそれぞれが一片の完結した知識であるため，その意味の理解が容易である．

(3) Self-Explanatory: 処理の全過程がWMの内容を順次変化させたルールの系列としてわかるので，いかにして結論が得られたかが，それを見れば容易に理解できる．

以上の3つの利点は知識工学の分野における情報処理のように，全体の制御の構造が不明確な分野，あるいはそれを事前に決定することが極めて困難な分野には望ましい特徴である．

それにもかかわらず，PSには効率の悪さ，制御構造の一様性に起因するプログラミングの困難さ等の欠点が指摘されてきた[6]．これ等の欠点を克服するために多くの研究[6][7]がなされているが，ほとんどのものは，タグ・マーカー等の特殊記号の導入に代表されるad-hocな改良であり，PSの優れた特性を犠牲にすることによりなされていた．

本報告ではPSの前述の3つの特徴を保存しつつ，さらにPSの能力を向上させるために基本PSを2方向の階層構造をもたせて配列することにより，新しく階層的プロダクションシステムHIPS (HIerarchical Production System)を設計し，インプリメントして良好な結果が得られたので報告する．

次節ではHIPS設計の基本思想及びシステムの概略について述べる．3節では基

本PSの記述及びバックトラック機構について，4節ではインプリメンテーションについて述べ，5節ではプログラム例とその実行例を示す．

## 2.システムの概観

### 2.1 概念的説明

PSを用いた知識表現に基づく問題解決システムを設計する際には，問題領域(problem domain)をほとんど独立した状態に分解し，問題固有の知識をその使われ方とは無関係にシステムに組み込まなければならず，通常システムの能力はその分解の結果に大きく依存する．従って，問題領域を適切に分解することは能率的な問題解決システムの構築にとって重要な問題となる．

音声理解システムHearsay-II[8][9]では問題領域を句，単語列，単語，音節，セグメント，パラメータ・レベルの6つのレベルに分けられている．又，Kanade[10]は画像理解システムに複数レベル表現(対象，副画像，領域，区画，画素レベル)を導入している．SobwayとRisemanの学習システム[11]には9レベルのパターン記述が導入されている．彼らのシステムでは学習は莫大な量の知識を必要とする外部世界のことを取り扱うための知識駆動型の解釈と考えられる．問題領域のこれらの複数レベルの表現は人間の直観にとって自然であり，そしてそれらはシステム設計者が問題をトップダウンに考えるのに役立っている．

GPS(General Problem Solver)の例では，抽象空間の概念がSacerdoti[12]によって用いられている．彼のABSTRIPSシステムにおいてプラン作成は次のようになされている．最初それは詳細な状況を無視して大まかなプランをつくる．この場合，最初のプランは最も抽象度の高い空間における一連の動作からなる．それは直接問題領域に適用できないかもしれないが，主要な因果関係の連鎖を損なってはいない．次に，一段低いレベルでの知識を使ってプランを詳細化する．この過程は実際に実行可能な動作の系列が得られるまで続けられる．

以上の様に，問題領域及び知識の階層的な(複数レベルの)表現は複雑な問題を取り扱う際には有益であることが分る．

PS設計に関する諸問題の中で，conflict resolutionは主要な課題である．Davis[1]はメタ・ルールの導入によってconflict resolution問題を論じている．メタ・ルールはconflict set内から次に実行すべきルールを選択するためのヒューリスティックなルールで，下位レベルの知識をいかにして使うかに関する知識と考えられる．さらに，メタ・メタ・ルールも考えられる．このルールの階層性は表現の一様性を保存しながら，ヒューリスティックな知識をシステムに組み込むことを可能にする．GoldsteinとGrimson[13]そしてHayes-RothとLesser[8]は同様の問題を考察している．それぞれ表現は異なっているが各recognize-act cycleにおいて付随したヒューリスティックな知識に基づき，より適したルールを発火させることによってシステムの能率を上げることを意図している点は同じである．

知的システムの構築に必要な以上の2種類の階層的知識を考慮して我々はFig.1にブロック・ダイヤグラムで示す階層的プロダクションシステムの一般的概念を得る．

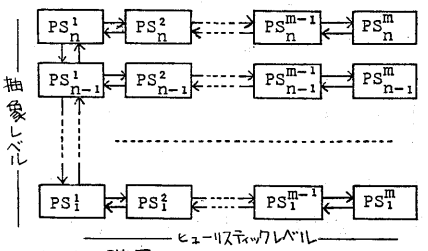

Fig.1 階層的プロダクションシステムHIPS$^m_n$
(各ブロックが1個の基本PSに相当する.)

HIPSは次の4つの特徴を持っている．

(1)縦の次元はプランの為の抽象レベルにあたる，問題に固有な知識の階層性に相当する．

(2)横の次元はconflict resolutionのための戦略としてのメタ・ルールの階層性に相当する．

(3)各構成要素は通常のPS構造を持っている．

(4)構成要素間の相互作用は隣同士に限定される．

HIPSはルール(知識)の階層構造を反映することによって既存のPSを自然な方向に拡張するように設計されており，そこではconflict resolutionのためのad-hocな手法は何も使われていない．我々のシステムではユーザが定義したヒューリスティックスを他の知識と同様にルールの形式で組み込むことができる．さらに問題固有の知識の階層性はシステムが各状態に適したルールの呼び出しをするのに役立つ．なぜならあるレベルでのPSはどのルールを発火すべきかを決定する時には抽象レベルにおいて次の一段高いレベルで設定されたサブゴールを参照することができるからである．このように，各レベルでのルールの呼び出しは，次の高いレベルのプランによって導かれる．

## 2.2 システムの動作の概略

ここではHIPSの縦と横のそれぞれのレベルにおける階層的な処理の概略を述べる．最初ユーザはシステムへ問題記述として初期状態(WMとなる)と最終(ゴール)状態，及び各階層に対応したルールすなわち階層的な問題固有の知識とヒューリスティックな知識のリスト，PMを与える．尚，HIPSの縦，横の各レベルはユーザが作成するこのPM内のルールの階層性の数に従って実行時に自動的に生成される．

Fig.1に基づいて説明する．まず最初HIPSは最も高い抽象レベルのルールを使って基本PSである$PS^1_n$を働かせ抽象度の高い解(初期状態と最終状態を結ぶ状態の系列)*を得る．しかし，その途中の各recognize-act cycleにおいてconflictが生じれば，ヒューリスティックレベルで1つ上のレベルの$PS^2_n$を起動し，conflict resolutionを行なう．** その際にヒューリスティックス用のルールがヌconflictを起こせば，さらに上位の$PS^3_n, PS^4_n, \cdots, PS^m_n$とconflictが完全に解消されるまで同様の操作が繰り返され，その後もとの$PS^1_n$でのrecognize-act cycleに戻り，処理を再開する．そうして最終的に解の系列が求まれば，それをもって一つ抽象レベルの低い$PS^1_{n-1}$に降り，そのレベルでのルールを使ってPS¹ₙで得られた解系列をサブゴール系列とみなし各サブゴール間を埋めていく．同様に横のヒューリスティックレベルのPSをも必要な際には，それを起動しながら抽象レベルを$PS^1_{n-2}, PS^1_{n-3}, \cdots, PS^1$へと徐々に降りていき，プランをより詳細なものへと具体化していく．

以上がHIPSの全体の動作の概略であるが，この間にWMとルールのLHSとのマッチングの際のバックトラック，あるrecognize-act cycleにおいて処理を進めることが出来なくなった際のバックトラック，及びある抽象レベルで得た解の中のサブゴールが適当ではないことが下位のレベルに降りて初めてわかった場合上位レベルにバックトラックしてそのサブゴールを修正し直すレベル間のバックトラックという3種類のバックトラック機構があるが，これに関する詳細な説明は3.3に述べる．

# 3. システムの記述

ここではHIPSの核となっている基本PSについて述べる．

## 3.1 ルールの構文(format)と意味

この節では，ルールとWMの形式及びその意味を述べる．HIPSのルールは，

```
(RULE-ID (LHS)--->(RHS))
```

なる形式をとっており，一般にLHSは

```
((B =X))
((C #X)(=X *))
(-(=X)(D =Y (=X #Y)))
```

などのように定数と"="又は"#"で始まる文字アトムである変数とのリストのリストである．"="で始まる変数は1個の，"#"で始まる変数は1個以上の任意の文字アトムにマッチし，各々マッチしたアトム及びアトムのリストと対応づけられる．1つのルール内の同じ変数には同じ値がマッチするようになっている．又，"*"はdon't careマークで何にでもマッチする

* 通常の意味での解は初期状態を最終状態へ変換するルールの系列として表わされるが，ここでは便宜上状態の系列として表わすことにする．

** ヒューリスティックレベルで上位のPSにおいては動作は主にWM(下位でのconflict set)の順序を入れ替える操作から成る．

が，値は不定である．そのため同一ルール内に"*"が現われても，そのたびに同じ値であるとは限らない．又，リストの前に"-"があるものは他のリストとは逆にこのリストがWM中に存在しない時にのみマッチする．尚，変数と値との結合はLHSをWMとマッチングする際に作ったA-LISTを介して，そのルールのRHSが実行される時に行われる．

以上がLHSの一般形式であるが，これら以外にも5節の例で示す(*DIFFERENCE)のようにユーザ定義のマッチ関数もあり，これは頭に"*"を付けてユーザが場合に応じて用意する．

一方，RHSは以下のようにWM中の要素を除去したり付け加えたり，WMの現在の状況をユーザに知らせたりするシステム関数からなる．

```
RHS system functions:
(DELETE arg)----- WMからargを除去する.
(DEPOSIT arg)--- WMにargを付け加える.
(SAY)------------- WMの現在の内容を知らせる.
(STOP)------------ 処理を終了する.          etc.
```

ここで"arg"は$(arg_1, arg_2, \cdots, arg_n)$のように複数個の引数を意味する．
ユーザはLHSと同じようにRHSにも5節の例で示す(REPLACE)などのようなRHS関数を用意できる．

以上述べた様にルールに現われる変数はルールに固有の局所変数であり，右辺には任意の関数を書くことができ，ルール間の干渉はWMを通してのみ行われることから各ルールはモジュール化が徹底された1つのプログラムとみることができる．

最後にWMは文字アトムからなる任意の構造を持つリストのリストである．例えば，

```
((A X C)(B X))
((1 5 (2 4)) (6) (9 8 2))
```

というような形式をとる．

## 3.2 Conflict Set

3.1に示した取り決めに従ってPSIはルールのLHSをWMの状況と照合し，起動条件が満足されているルールの集合conflict setをつくる．

次の様なWM及びPMが与えられた場合，

```
WM:
  ((A B C)(A C D)(B X)(E D A)(A)(A E C))
PM:
  ((P1 ((A =X C)(B X)-(B E))--->((DELETE (B X))))
   (P2 ((B X)(A =X *))--->((DEPOSIT (A =X B))))
   (P3 ((=X)(#Y =X)-(C =X))--->((SAY))))
```

conflict setは以下の如くになる

```
Conflict Set:
  ((P1 ((DELETE (B X)))((=X.B)))
   (P1 ((DELETE (B X)))((=X.E)))
   (P2 ((DEPOSIT (A =X B)))((=X.B)))
   (P2 ((DEPOSIT (A =X B)))((=X.C)))
   (P2 ((DEPOSIT (A =X B)))((=X.E)))
   (P3 ((SAY))((=X.A)(#Y.(E D)))))
```

例えばルールP1に対しては変数=Xに値B，又は値Eを対応づけることによってWMとマッチする．そしてあらかじめユーザにより用意された戦略に従って横のヒューリスティックレベルでこの6つの中から1つのルールを選択し，実際にそのルールのRHSに示されたシステム関数をWMに対して解釈実行していく．尚，ヒューリスティックレベルでのルールがユーザによって用意されていない場合には，システムはconflict set内の先頭のルールから順に選択するようになっている．

## 3.3 バックトラック機構

HIPSの全処理中には前述の如く3種類のバックトラックが必要に応じて行なわれる．以下に各々について述べる．

### 3.3.1 マッチングのバックトラック

このバックトラックはWMの内容とルールのLHS中の各要素をマッチングする際に行なわれる．例えば簡単な例で示すと，

```
((B A C)(B C)(C A B))
```

というWMに対して，

```
((=X A =Y)(B =X))
```

なるルールのLHSが与えられた場合，PSIは前から順に走査していくため，最初=Xと=Yという変数に関して，それぞれ値BとCとの結合が行われるが，次の要素(B =X)の変数=Xに値Bを代入して(B B)としても，このような要素はWM中に存在しない．そこで(=X A =Y)の最初の要素にバックトラックし，他の変数のbinding(結合)の可能性を探索する．その結果，変数=Xは値C，=YはBと結合され，次の要素(B =X)は今度は(B C)となりマッチングは成功するという具合である．しかし実際にはルールのLHS中の要素の数や構造，及び変数の種類や数が多いため，より複雑なバックトラックが幾度となくなされ

た後にマッチングは成功することになる．

### 3.3.2 recognize-act cycle のバックトラック

このバックトラックはあるrecognize-act cycleにおいて，その時のWMに対して適用するルールが1個も存在しなかったり，過去に幾度か繰り返してきたrecognize-act cycleにおけるWMと同じものになった際ループを防ぐため一つ前のrecognize-act cycleに戻り，その際のWMに対してその際のconflict set内の2番目のルールを実行することによって行なわれる．今，

```
((P1 ((A)(B))--->((DELETE (B))))
 (P2 ((A)-(B)-(C))--->((DEPOSIT (B))))
 (P3 ((A)-(C))--->((DEPOSIT (C)))))
```

というPMに対して，

```
((A)(B))
```

なるWMが与えられた場合を考える．

1st recognize-act cycleでは，conflict set:(P1 P3)
この場合ヒューリスティックレベルでのルールは与えられていないので先頭のP1が実行され，その結果WMは((A))となる．

2nd recognize-act cycleでは，conflict set:(P2 P3)
同様にしてP2が実行され，WMは((A)(B))となる．しかし，このWMは最初に与えられたWMと全く同じであるためループしてしまう．故に2nd recognize-actcycleで見つけたconflict set内の2番目のルールP3が実行される．その結果，WMは((A)(C))となる．

3rd recognize-act cycleでは，
WM:((A)(C))に対して，どのルールも発火しない．それで2nd recognize-act cycleにバックトラックし，conflict set内の次のルールを実行し直そうとするが，既に未実行のルールは残っていないので，さらに1st recognize-act cycleにバックトラックし，WMをその時の((A)(B))に戻しconflict set内の2番目のルールP3を実行する．その結果，WMは((A)(B)(C))となる．そして，このWMに対して同様にrecognize-act cycleを繰り返していくという具合である．

しかし，このrecognize-act cycleのバックトラックが度々行なわれると非常に効率が悪くなる．従って，そのためにHIPSにはヒューリスティックレベルの階層性が用意されており，そこにユーザがヒューリスティックルールをかくことができる．従って，そのルールを使ってconflict set内のルールを優先順位の高いものから低いものへと再配列してルールの適切な選択が行なえるので実際にはこのバックトラックの回数を少なくすることができ，効率の良い処理が可能となる．

### 3.3.3 レベル間バックトラック

このバックトラックはある抽象レベルでの解であるサブゴールの系列の内のあるサブゴールが次の一つ下位のレベルにおいて不適当であるとわかった場合，元の上位レベルにバックトラックして，その不適当なサブゴールを修正し直して再び下位レベルの処理を続けるというものである．Fig.2を使って簡単に説明

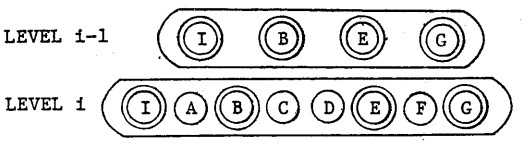


Fig.2 抽象レベルでのバックトラック

する．今，LEVELi-1で得た解の系列Ⓘ→Ⓑ→Ⓔ→Ⓖを LEVELiでもう少し具体化して，解の系列がⒾ→Ⓐ→Ⓑ→Ⓒ→Ⓓ→Ⓔ→Ⓕ→Ⓖとなったとする．ただし，この場合PSはⒾ-Ⓑ間，Ⓑ-Ⓔ間，Ⓔ-Ⓖ間の3回働いている．そして次にLEVELi+1に降りて，さらに具体的な解を得るためPSをⒾ-Ⓐ間，Ⓐ-Ⓑ間，Ⓑ-Ⓒ間と働かせている間にⒸ-Ⓓ間をつなぐルールの系列が1つも存在しなかった場合にはLEVELiにバックトラックし，Ⓒ-Ⓔ間をつなぐサブゴールの系列を他にないか探索し，再びLEVELi+1でサブゴールⒸからの処理を続ける．又Ⓒと新しいゴールを結びつけることが出来なければ再びLEVELiにバックトラックし他のサブゴールを探すがⒸ-Ⓔ間をつなぐ経路が他になくなると今度はⒷ-Ⓔ間をつなぐ他の経路を探しLEVELi+1で処理を再開するが，それも失敗すれば最後にはさらにLEVELi-1にバックトラックしLEVELiで見つけたサブゴールⒺをも変更してみる．同様にして，このバックトラックはさらに上位のレベルに伝搬し得るが，LEVELi-1の上位がもうⒾとⒼだけの初

期状態の場合には解なしということで処理は終了する．

## 4. インプリメンテーション

HIPSはLisp1.9を用いてインプリメントされ，作業領域を含めて80KB程のシステムである．我々の主な目的は知的システム構築用のプログラミング言語を開発し，その使用経験を通してPS自体の有効性を確認することにあるので，スピード及び効率については現在のところ考慮を払っていない．

### 4.1 基本方針

HIPSのインプリメンテーションの方針は，知識の各抽象レベル上での動作を全て基本PSによって統一して表現しようとする点に，その基礎を置いている．従って個々のPSが全く同一の制御構造を持っており，全体の制御の流れが常に一様であることから，Fig.3のように一個の基本PSをまず作り，再帰的に働かせることによって，ヒューリスティックレベル全体のPSの階層性をインプリメントした．又，縦の抽象レベルについても横のヒューリスティックレベル全体を再帰的に呼び出すことによって全体としてFig.3のような階層性を持ったPSに拡張した．この様にレベル間バックトラックの内部の詳細な処理以外では全て再帰構造を用いてバックトラック機構をインプリメントしたのでプログラムは，かなりコンパクトでかつ見易いものになった．

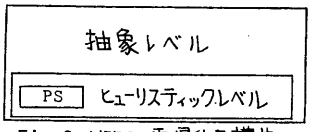

Fig.3 HIPSの再帰的な構造

### 4.2 概略

```
PROCEDURE <HIPS>;
BEGIN
  READ (initial state, goal state and PM);
  <abstract level>;
END

FUNCTION <abstract level>: BOOLEAN
BEGIN
  IF PM is empty THEN RETURN with success;
  BEGIN
LOOP:CASE <one-level loop> OF (*using rules of (CAR PM)*)
       success:
         CASE <abstract level> OF (*using rules of (CDR PM)*)
           success:RETURN with success;
           failure:BEGIN
                     search the inappropriate subgoal;
                     GOTO LOOP (*search for another possibility*)
                   END
         END
       failure:RETURN with failure (*level backtrack*)
    END
  END
END

FUNCTION <one-level loop>: BOOLEAN
BEGIN
LOOP:IF there is only one subgoal
       THEN RETURN with success
       ELSE BEGIN
              WM:=first subgoal;
              goal:=second subgoal;
              CASE <recognize> OF
                success:BEGIN
                          delete first subgoal;
                          GOTO LOOP
                        END
                failure:RETURN with failure;
              END
            END
END

FUNCTION <recognize>: BOOLEAN
BEGIN
  IF WM matches the goal state THEN RETURN with success;
  IF WM matches one of the old states
    THEN BEGIN
           erase the current state;
           RETURN with failure (*backtrack*)
         END;
  REPEAT (*make conflict set*)
    IF LHS of a rule matches WM
      THEN add the rule to conflict set
  UNTIL PM is exhausted;
  <conflict resolution>;(*rearrange the rules of conflict set
                          according to a certain strategy*)
  RETURN <act>
END

FUNCTION <act>: BOOLEAN
BEGIN
LOOP:IF conflict set is empty
       THEN RETURN with failure (*backtrack*)
       ELSE BEGIN
              execute RHS of the first rule of conflict set;
              delete the first rule from conflict set;
              CASE <recognize> OF
                success:RETURN with success;
                failure:BEGIN
                          set WM to the last one;
                          GOTO LOOP
                        END
              END
            END
END

PROCEDURE <conflict resolution>;
BEGIN
  IF conflict set contains more than one rule
    THEN <recognize> (*but using rules for conflict resolution*)
END
```

インプリメンテーションの概略をパスカル風に記述すると以上のようになる．

## 5. HIPSの実行例

本節ではロボット・ハンドによるブロックの問題に関して作成したルール及びその実行結果を用いてHIPSの動作について述べる．我々の問題の初期状態と最終状態をFig.4に示す．WMにおけるブロック及びロボット・ハンドの状態の表現方法を以下の如く定めた．

(ONTABLE X)……ブロックXがテーブル上にある状態．

(CLEAR X)……ブロックX上に他のブロックが

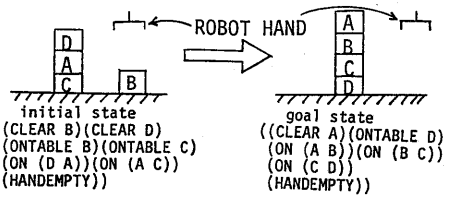

Fig.4 ロボットハンドによるブロックの問題

ない状態.

(ON (X Y))……ブロックXがブロックY上にある状態.

(HOLDING X)……ロボット・ハンドがブロックXをつかんでいる状態.

(HANDEMPTY)……ロボット・ハンドが何もつかんでいない状態.

従って，Fig.4の初期状態及び最終状態の場合にはWMの内容はそれぞれの図の下に示されたようなリスト構造で表現されることになる．次に示すルールのリストPMを作成した．

```
(((N1 ((CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (ONTABLE A) (ONTABLE B)
       (ONTABLE C) (ONTABLE D) (HANDEMPTY))---->
      ((*SUBGOAL ((CLEAR A) (ONTABLE D) (ON (A B)) (ON (B C)) (ON (C D))
                 (HANDEMPTY))) (SAY)))
 (N2 ((*)) ----> ((*SUBGOAL ((CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D)
                            (ONTABLE A) (ONTABLE B) (ONTABLE C) (ONTABLE D)
                            (HANDEMPTY))) (SAY))))
 ((P1 ((ONTABLE =X) (CLEAR =X) (CLEAR =Y)) ---->
      ((DELETE (ONTABLE =X) (CLEAR =Y))
       (DEPOSIT (ON (=X =Y)) (SAY)))
  (P2 ((ON (=X =Y)) (CLEAR =X) (CLEAR =Z)) ---->
      ((DELETE (ON (=X =Y)) (CLEAR =Z))
       (DEPOSIT (ON (=X =Z)) (CLEAR =Y)) (SAY)))
  (P3 ((ON (=X =Y)) (CLEAR =X)) ---->
      ((DELETE (ON (=X =Y)))
       (DEPOSIT (ONTABLE =X) (CLEAR =Y)) (SAY))))
 ((Q1 ((*DIFFERENCE)) ----> ((REPLACE) (STOP))))
 ((R1 ((ONTABLE =X) (CLEAR =X) (HANDEMPTY)) ---->
      ((DELETE (ONTABLE =X) (CLEAR =X) (HANDEMPTY))
       (DEPOSIT (HOLDING =X)) (SAY)))
  (R2 ((HOLDING =X)) ---->
      ((DELETE (HOLDING =X))
       (DEPOSIT (ONTABLE =X) (CLEAR =X) (HANDEMPTY))
       (SAY)))
  (R3 ((HOLDING =X) (CLEAR =Y)) ---->
      ((DELETE (HOLDING =X) (CLEAR =Y))
       (DEPOSIT (HANDEMPTY) (ON (=X =Y)) (CLEAR =X))
       (SAY)))
  (R4 ((HANDEMPTY) (CLEAR =X) (ON (=X =Y))) ---->
      ((DELETE (HANDEMPTY) (CLEAR =X) (ON (=X =Y)))
       (DEPOSIT (HOLDING =X) (CLEAR =Y)) (SAY))))
 ((S1 ((*DIFFERENCE)) ----> ((REPLACE) (STOP)))))
```

ここでN1～N2，P1～P3，R1～R4は各々抽象レベル1，抽象レベル2，抽象レベル3におけるルールである．又，Q1及びS1は各々抽象レベル2及び3でのヒューリスティックレベル用のルールである．抽象レベル1でのヒューリスティックレベル用のルールは用意されていない．LEVEL3における4つのルールは以下のような実際のロボットハンドの動作に対応し，

pickup(X)…テーブル上にあるブロックXを持ち上げる動作(R1).

putdown(X)…ブロックXをテーブル上に降ろす動作(R2).

stack(X,Y)…ブロックXをブロックY上に置く動作(R3).

unstack(X,Y)…ブロックY上にあるブロックXを持ち上げる動作(R4).

LEVEL2における3つのルールはどのブロックを移動させるかをロボット・ハンドのことは気にせずに決定するための知識に対応する．そしてLEVEL1での2つのルールは，そこを通れば早く最終状態に到達できるとユーザが考える状態を(*SUBGOAL)というRHS関数を用いてサブゴールとして設定する働きをする．又，LEVEL2と3におけるヒューリスティックレベル用のルールのLHSである(*DIFFERENCE)はconflict set内のルールをすべて実行した場合に，それぞれの結果とサブゴールとの近さを評価する関数であり，RHSの(REPLACE)はその評価値の大きいもの順にconflict set内のルールを再配列する働きをする．LEVEL1におけるルールは，抽象化されたルールというよりはむしろヒューリスティックスに近い意味を持っていることが分る．

前述のPM及び初期状態，最終状態を入力した結果，下に示すようなrecognize-act cycleを繰り返し，Fig.5のような解の系列が求まった．

```
**************************************
(1.)

CONFLICT-SET:
①((R1) ((DELETE (ONTABLE =X) (CLEAR =X) (HANDEMPTY)) (DEPOSIT (HOLDING =X
)) (SAY)) ((=X . A)))
 ((R4) ((DELETE (HANDEMPTY) (CLEAR =X) (ON (=X =Y))) (DEPOSIT (HOLDING =X
 ) (CLEAR =Y)) (SAY)) ((=Y . C) (=X . B)))

 ###CONFLICT-RESOLUTION-START###

②(<ONE-RULE-IS-FIRED>
 ((S1) ((REPLACE) (STOP)) NIL)

 (THE RULE S1 IS SELECTED)

③(CRITICALITY-VALUES:
 (5. 5.)

 ###CONFLICT-RESOLUTION-IS-DONE###

 <NOW-THE-CONFLICT-SET-IS-ARRANGED-AS-FOLLOWS>
④(((R4) ((DELETE (HANDEMPTY) (CLEAR =X) (ON (=X =Y))) (DEPOSIT (HOLDING =X
 ) (CLEAR =Y)) (SAY)) ((=Y . C) (=X . B)))
 ((R1) ((DELETE (ONTABLE =X) (CLEAR =X) (HANDEMPTY)) (DEPOSIT (HOLDING =X
 )) (SAY)) ((=X . A)))

⑤ (THE RULE R4 IS SELECTED)

 >>>THE-CURRENT-STATE-DESCRIPTION<<<
 ((CLEAR C) (HOLDING B) (ON (C D)) (CLEAR A) (ONTABLE A) (ONTABLE D))

⑥ THIS-STATE-OCCURS-ON-THE-PATH-BACK-TO-THE-INITIAL-STATE?!

⑦ (THE RULE R1 IS SELECTED)

 >>>THE-CURRENT-STATE-DESCRIPTION<<<
 ((HOLDING A) (ON (B C)) (ON (C D)) (CLEAR B) (ONTABLE D))

 **************************************
```

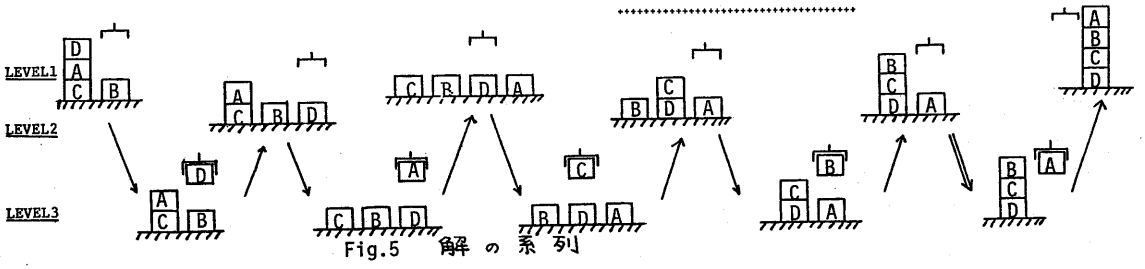


Fig.5 解の系列

この例はFig.5の⇒に対応するrecognize-act cycleの一例である．①に示すように(pickup A)と(unstack B)の2つのルールがconflictしたので，conflict resolutionのPSが生成され，②に示すルールS1が発火される．S1はR1とR4とを仮に実行してみてその結果がゴールに近い順に再配列しようとするが，この場合近さが等しかったので(③)，一応④に示すように再配列し，conflict resolutionが終る．そして⑤においてR4が実行されるが，その結果は以前に一度通った状態と同じであるので⑥)，バックトラックし，⑦でconflict set内の次のルールR1が実行される．

ところでLEVEL 1でのサブゴールの設定をしなかった場合には，LEVEL 2においてFig.6のような遠まわりの解の系列が求まった．このように抽象レベルでの階層性を高くしてサブゴールの与え方を適切に行なうことは優れた解を得る為には重要であることが分る．

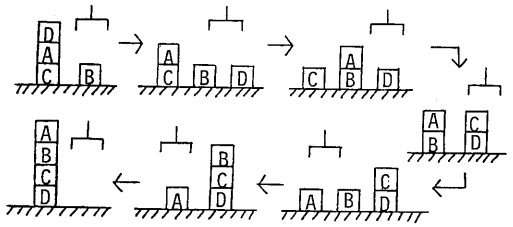


Fig.6 LEVEL 1でsubgoalの設定をしなかった場合に得られるsubgoalの系列

## 6. おわりに

階層的プロダクションシステムHIPSを紹介し，その実行例を示した．HIPSは2次元の階層構造をもつ複数の基本PSから構成され，人間が持っているような問題固有の階層的知識，及びそれ等の知識の使い方に関するヒューリスティックな知識を全てプロダクションルールの形で表現することを可能にしている．HIPSは既存のPSの自然な拡張になっており，プログラミング(ユーザが持つ知識の埋込み)がかなり容易になるものと期待される．

尚，知識駆動型のシステム構築への応用を現在検討中である．


## 謝辞

最後にHIPSのプログラミング，及び本報告作成にあたり貴重な援助をいただいた大阪大学産業科学研究所角所研究室大学院生富田雅巳君に深く感謝致します．


## 参考文献


[1] Davis, R. and J. King: "An overview of production systems", Machine Intelligence, 8, pp. 300-332 (1977).
[2] 辻井潤一，「プロダクションシステムとその応用」，情報処理 Vol.20, No.8, pp. 735 - 743 (1979).
[3] 佐藤泰介，「プロダクションシステムの試作とその使用経験」，情報処理学会，人工知能と対話技法研資3-1 (1978).
[4] Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg: "On generality and problem solving —— A case study involving the DENDRAL program", Machine Intelligence, 6, pp. 165-190 (1971).
[5] Shortliffe, E. H.: "MYCIN: A rule-based computer program for advising physicians regarding antimicrobial therapy selection", Stanford Univ. Computer Science Report, CS-74-465 (1974).
[6] Rychener, M. D.: "Control requirements for the design of production system architectures", Proc. of the Symposium on Artificial Intelligence and Programming Languages, pp. 37-44 (1977).
[7] Moran, T.P.: "The symbolic imagenary hypothesis: A production system model", Computer Science Department, Carnegie-Mellon Univ. (1973).
[8] Hayes-Roth, F. and V. R. Lesser: "Focus of attention in the Hearsay-II", Proc. of the 5th IJCAI, pp. 27-35 (1977).
[9] Erman, L. D. and V. R. Lesser: "A multilevel organization for problem solving using many, diverse, cooperating sources of knowledge", Proc. of the 4th IJCAI, pp. 483-490 (1975).
[10] Kanade, T.: "Model representations and control structures in image understanding", Proc. of the 5th IJCAI, pp. 1074-1082 (1977).
[11] Soloway, E. A. and E. M. Riseman: "Levels of pattern description in learning", Proc. of the 5th IJCAI, pp. 801-811 (1977).
[12] Sacerdoti, E. D.: "Planning in a hierarchy of abstraction spaces", Artificial Intelligence, 5, pp. 115-135 (1974).
[13] Goldstein, I. P. and E. Grimson: "Annotated production systems —— A model for skill acquisition", Proc. of the 5th IJCAI, pp. 311-320 (1977).