北海道大学工学部研究報告
第108号（昭和57年）

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 108 (1982)

適応プロダクションシステム(APSH)の作成

小林　茂*　桃内佳雄　宮本衛市

（昭和56年12月26日受理）

Construction of Adaptive Production System (APSH)

Shigeru KOBAYASHI, Yoshio MOMOUCHI and Eiichi MIYAMOTO

(Received December 26, 1981)

Abstrat

In this paper, we describe about an adaptive production system APSH.

APSH has the ability to create new production rules, to remove production rules, and to deposit conditions and actions of production rules into the working memory. Operations to existing production rules use the information of the right hand side of production rules, and these operations are important functions of the system.

APSH is also constructed as a flexible programming system. For example, it allows for a description of a compositive program which includes some production memories, and to edit programs within the system.

APSH is expected to be a useful tool for some studies in cognitive science, such as programming of models of learing or problem solving.

The ability and capability of the system is discussed with an example program, which predicts the succeeding part of a sequence of letters.

1. まえがき

プロダクションシステム (PS: Production System) は Newell と Simon によって人間の問題解決過程のモデル化のために提案され，最近，認知科学あるいは知識工学といった分野で多く用いられているプログラミング手法である。従来のプログラミング手法，たとえば PASCAL などによるプログラミングでは情報処理における処理の制御の構造が事前に明確に規定されていることが要求される。これに対して認知科学あるいは知識工学の分野における情報処理では処理の制御の構造が不明確，あるいは事前に規定することができないことが多く，従来のプログラミング手法は適用しにくい。PS はこのような分野に適したプログラミング手法である。

PS のプログラムは，プロダクションルール (PR: Production Rule) と呼ばれる条件と行為の対の集合として記述される。PR の行為は条件が満足されたときに実行される。PR はそれぞれが情報処理に関する完結した記述であり，PR 間の関係は弱いため，PR に操作を加えることによって比較的容易にプログラムを修正することができる。プログラムの実行によってプログラム

情報工学専攻　情報システム工学講座
* 現在，東京芝浦電気㈱勤務

自身を書き換えることのできるPSは適応PSと呼ばれる。本論文で報告するAPSHはPRの生成，削除，参照を行なう能力を持つ適応PSである。同時にAPSHでは，PRの集合を複数個用いたプログラムなど，複雑な制御構造を持つプログラムも記述できるよう考慮されている。これらの機能が従来のPSよりも拡張されており，APSHは，認知科学の分野における重要な問題である学習，問題解決のモデル等のプログラミングのための有用な道具となると考えられる。

2. APSHの機能

2.1 PSの構成と機能

PSの基本的な構成を図1に示す。PSはプロダクションメモリ(PM: Production Memory)，ワーキングメモリ(WM: Working Memory)およびインタープリタを基本構成要素とする。PMはPRの集合，WMは順序づけられた記憶要素(ME: Memory Element)の列である。PRは“C ⇨ A”の形で表わされる。ここでCは条件要素(CE: Condition Element)の並び，Aは行為の並びで，それぞれ左辺(LHS: Left Hand Side)，右辺(RHS: Right Hand Side)と呼ばれる。PRのLHSは条件を表わし，LHSが満足されるとそのPRのRHSの行為が左から順に実行される。PMがPSにおけるプログラムである。インタープリタは各PRのLHSとWMを比較し，LHSの満足されるPRを実行する。複数個のPRのLHSが同時に満足されることもあるので，インタープリタはそれらの中からただ1つのPRを選択するための規則を持っている。PRの行為はWMに対して働きかけるものが多い。1つのPRの実行が終了すると，新しいWMの内容に対して次のPRが選択，実行される。PSは次のような特長を持つ[1]。

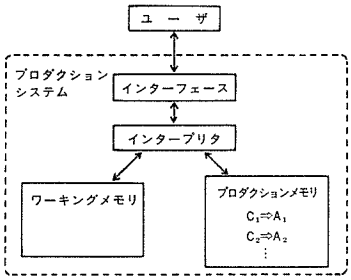

図1 PSの構成

① PR間の相互干渉はWMを通して間接的に行なわれるので，PRの独立性が強い。

② プログラムに用いられている知識は，通常のプログラムでは処理の流れの中に埋もれているため理解するのが困難だが，PSではPRという完結した知識の集合として表わされるので明確である。

PSのこの特長はプログラム自身によるプログラムの修正を容易にする。PRをデータと見なし行為によって操作を加えることのできるPSは適応PSと呼ばれる。PRに対する操作として考えられるものには，新しいPRを生成してPMに加えること，既存のPRの一部を修正すること，あるいはPMから削除すること等がある。WatermanによるPAS-II[2]は，誤りを生じるPRの前に正しいPRを付加し，それを優先的に実行することによってPM全体として正しいプログラムを表現するようにPMを修正して行く適応PSであり，PRの生成のみによっても適応的機

能の実現が可能であることを示している。しかし，APSH では PR の生成のほかに，既存の PR を削除すること，PR をデータとして WM に加え参照することができるようにした。これによって，不要な PR の削除や既存の PR の一部を修正した PR の生成が可能になると考えられる。

2.2 APSH の構成

APSH の構成を図 2 に示す。APSH のプログラムはユーザが定義した行為の集合である。ユーザが定義する行為には行為の列として定義されるものと，PR の集合として定義されるものがあり，それぞれ「定義行為」，「手続き」と呼ばれる。1 つの手続きが，基本的な PS の PM に相当する。これらに対し，システムで予め定義されている行為は「基本行為」と呼ばれる。APSH はユーザからのコマンドの入力とその実行を反復する。基本行為，ユーザが定義した行為はいず

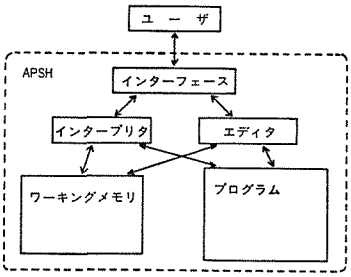

図 2 APSH の構成

れもコマンドとして使用することができる。ユーザが定義した行為はコマンドとして実行される場合と，他の行為の定義の中に用いられ，呼び出されることによって実行される場合がある。プログラムをこのように行為の集合として記述すると，基本的な PS の，ただ 1 つの PM によるプログラムに比べ次のような利点がある。

① 階層構造を持つ PM，逐次的に実行される PM の集合などの構成が自由に行なえる。

② まとまった作業に名前をつけ，サブプログラム化することにより，プログラムの記述，解読が容易になる。

③ PM を分割することにより，実行すべき PR の選択に要する計算時間を短縮できる。

また，APSH はプログラムの作成，修正を容易にするためにエディタを内蔵している。

2.3 行為の定義形式

定義行為，手続きはそれぞれ次の形式に従って定義される。

① 定義行為

（（定義行為名 [仮引数$_1$ 仮引数$_2$ …]）＝（行為$_1$）[（行為$_2$）…]）

② 手続き

（手続き名 [仮引数$_1$ 仮引数$_2$ …]）

$$\begin{bmatrix} \text{プロダクションルール}_1 \\ \text{プロダクションルール}_2 \\ \vdots \end{bmatrix}$$

ここで，[ ] はその内容が必須でないことを表わす。行為$_i$ は行為名と適当な引数の並びである。PR は次の形で表わされる。

（ルール名 [条件$_1$] [－条件$_2$] ⇨ （行為$_1$）[（行為$_2$）…]）

ルール名はその PR を表わす整数，条件$_i$ は CE の並びである。LHS に “－” が含まれるとき，

その左側にある CE を「正の CE」，右側にある CE を「負の CE」と呼ぶ。LHS は正の CE，負の CE のいずれか一方であってもよいし，空であってもよい。手続きの定義として PR を持たないものも認めているのは，後にプログラムの実行によって PR が加えられる場合があるためである。

2.4 定義行為，手続きの実行

定義行為の実行とは，“=” の右側の行為を左から順に実行することをいい，行為の列の最後まで実行が終了したとき，その定義行為の実行が終了する。

手続きの実行は LHS の満足される PR の選択とその実行の反復である。APSH では PR の並んでいる順序が PR の優先順序を表わし，複数の PR の LHS が同時に満足された場合にはそれらの中で最も先頭に近いものが実行される。PR の LHS は次の 2 つが成り立つとき満足される。

① 正の CE のそれぞれに対して，一致する ME が WM 中に存在する。

② いずれの負の CE に対しても，一致する ME が WM 中に存在しない。

CE と ME が一致するとは，CE の要素が ME の先頭から同じ順序で含まれることをいう。また CE は変数を含むことがある。変数は第 1 文字が “？” である文字列で表わされる。変数はどのような要素とも等しいと見なすことができるが，1 つの PR 内では同じ要素を表わす。図 3(a)は一致する CE と ME の例，図 3(b) は一致しない CE と ME の例である。1 つの ME は 1 つの CE に対してのみ一致するが，CE の並んでいる順序は ME の順序に無関係である。PR の LHS とそれを満足する WM の例を図 4(a) に，LHS を満足しない WM の例を図 4(b) に示す。

CE		ME
(A B)	⟷	(A B)
(A)	⟷	(A B)
(?X)	⟷	(A)
(A ?Y)	⟷	(A B)

図 3(a) 一致する CE と ME の例

CE		ME
(A)	⟷	(B A)
(A B)	⟷	(A C B)
(A ?X)	⟷	(A)
(?Y ?Y)	⟷	(A B)

図 3(b) 一致しない CE と ME の例

LHS		WM
(A)(B)	⟷	(B)(C)(A)
(?X)(?X B)	⟷	(A)(A B)
(?X)(?Y)	⟷	(A)(B)
(A)−(B)	⟷	(A)(C)

図 4(a) LHS を満足する WM の例

LHS		WM
(A)(A)	⟷	(A)(B)
(?X)(?X)	⟷	(A)(B)
(A)−(B)	⟷	(A)(B)
(A ?Y)−(A ?Y)	⟷	(A B)(A B)

図 4(b) LHS を満足しない WM の例

このほか特殊な条件として，正の CE の中に文字列の比較を含めることができる。この CE は

$(=\ a\ b)$, $(<>\ a\ b)$, $(<\ a\ b)$, $(<=\ a\ b)$

の 4 つの中のいずれかの形をとる。ただし，a および b は文字列または文字列を値とする変数である。最初の要素が “=” である CE は a と b が同じ文字列であるという条件，“<>” である CE は同じ文字列でないという条件を表わす。最初の要素が “<” または “<=” である CE では a および b は共に整数を表わす文字列でなければならず，それぞれ $a<b$，$a\leqq b$ が成立しなければならないという条件を表わす。このような CE を含む LHS の例を図 5(a)，図 5(b) に示す。

```
LHS : ( ?X)(<> A ?X)          <---> WM : (B)
(M ?Y)(< ?Y 5)                <--->      (M 2)
(A ?X)(B ?Y)(<= ?X ?Y)        <--->   (A 6)(B 7)
```

図5(a) LHSを満足するWMの例

```
LHS : ( ?X)(<> A ?X)          <---> WM : (A)
    (M ?Y)(< ?Y 5)            <--->      (M 9)
    (A ?X)(B ?Y)(<= ?X ?Y)    <--->      (A 7)(B 6)
```

図5(b) LHSを満足しないWMの例

2.5 PRに対する操作のための行為

PRの操作に関するAPSHの行為は，新しいPRの生成に関するものと，既存のPRの削除および参照に関するものの，2つのグループに分けることができる。

PRの生成に関する基本行為を表1に示す。

表1 PRの生成に関する基本行為

(COND c) : (COND c) をWMに加える。
(NCOND n) : (NCOND n) をWMに加える。
(ACTION a) : (ACTION a) をWMに加える。
(PROD p) : WM中から (COND c_i), (NCOND n_j), (ACTION a_k) の形をしたMEをすべて取り出し，c_1, c_2 …… を正のCE，n_1, n_2 …… を負のCE，a_1, a_2 …… を行為とするPRを作り，これを p という名前で定義されている手続きの先頭に加える。

たとえばWMの内容が (COND (S SPARROW)) (ANS BIRD) であるときに，PR

(2 (ANS ?X) ⇨ (ACTION (REPLY ?X)) (PROD PM1))

が実行されると，PM1という名前で定義されている手続きの先頭に新しいPR

(n (S SPARROW) ⇨ (REPLY BIRD))

が加えられる。生成されたPRのルール名 n は，PM1内でルール名として使われている整数のうち最大のものに1を加えた数が設定される。このPRの実行後，WMの内容は (ANS BIRD) となる。

PRの削除，参照に関する基本行為を表2に示す。

表2 PRの削除，参照に関する基本行為

(CHECK p [*] a) : 操作を加えるPRを指定する。指定されるPRは p という名前で定義されている手続きの中で，RHSが a に等しいPRである。第2引数に "*" を与えると，指定されるPRは実行されたことのあるPRの中から選択される。該当するPRが見つかればWMにME (RFOUND) を加える。
(REMRULE) : "CHECK" で指定されたPRを削除する。
(DISSECT) : "CHECK" で指定されたPRの正のCE c_i に対して (COND c_i)，負のCE n_j に対して，(NCOND n_j)，行為 a_k に対して (ACTION a_k) をWMに加える。これは "PROD" の逆の操作であるが元のPRは保存される。
(NOCHECK) : "CHECK" で指定されたPRを，以後 "CHECK" による指定の対象からはずす。
(RELEASE p) : "NOCHECK" によって "CHECK" の指定の対象からはずされた手続き p 内のPRを，指定の対象となり得るようにする。

"CHECK" によって操作を加える PR を指定する時，その PR を RHS によって識別するのは次の理由による。

① 操作を加えるべき PR の全体の形を知ることは一般に困難である。

② PR に操作を加える必要性が認められるのは，誤った行為が実行された場合，実行されるべき行為が実行されなかった場合など，行為が正しく働かなかった場合であることが多い。従って，これらの行為から，操作を加えるべき PR の RHS は推測可能なことが多いと思われる。

図6のように定義されている手続き PM1 内の PR に対する操作について考えてみよう。いま，この手続の外部で，行為 (CHECK PM1 (REPLY MAMMAL)) が実行されたとすると，RHS

```
(PM1)
(4 (S SPARROW)=>(REPLY BIRD))
(3 (S CAT)=>(REPLY MAMMAL))
(2 (S DOG)=>(REPLY MAMMAL))
(1 =>(RETURN))
```

図6 PM の例

(REPLY MAMMAL) を持つルール名 "3" の PR が操作の対象として指定される。この状態で行為 (DISSECT) が実行されると，WM の先頭には2つの ME (COND (S CAT)) (ACTION (REPLY MAMMAL)) が加えられる。また，ここで行為 (NOCHECK) を実行した後に再び (CHECK PM1 (REPLY MAMMAL)) が実行されると，操作を加える PR として，ルール名 "2" の PR が指定される。

3. APSHによるプログラムの例

次に，APSH 上で記述されたプログラムとその実行結果の例を示す。ここで与えた問題は図7(a)のように，アルファベットの列を与えられ，その後に続く文字列を予測するものである[2]。この予測を行うために次の2つの仮定を用いる。

① 与えられた文字列は，ある周期に基く規則性を持つ。

② その周期で文字列を区切ったとき，隣りあう各部分の中で相対的に同じ位置にある文字は常に同じ関係にある。

ここでいう文字と文字との関係とはアルファベットとしての順序の差である。この関係を満足する文字列の例を図7(b)に示す。この仮定を用いたアルゴリズムは次のように書くことができる。

(i) 周期を1と仮定する。

(ii) 与えられた文字列の最初の2周期分を取り出し，相対的に同じ位置にある各文字間の関係を求める。

```
FAFBFC  ⟶ FD
CCCBBB  ⟶ AAA
ABABA   ⟶ BA
ABCABD  ⟶ ABE
ABCBCD  ⟶ CDE
```

図7(a) 文字列の予測の例

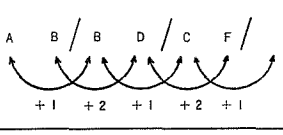

図7(b) 周期的に同じ関係を持つ文字列の例

(iii) 求めた関係を文字列の3周期目以降の部分に適用する。

(iv) 与えられた文字列を最後まで調べ，矛盾が生じなければ後続部分を予測して終了する。

(v) 途中で矛盾が生じたら，仮定した周期を1だけ増加させ，(ii)へ戻る。

図8はこのアルゴリズムに基くプログラムである。その中で用いられている基本行為の実行内容を表3に示す。（表1，表2に既出のものは除く。）

表3 基 本 行 為

(CLEAR *a*)	: 第1要素が *a* でない ME をすべて消去する。
(REM #*n*)	: #*n* の指す ME (PR の *n* 番目の CE に一致した ME) を消去する。
(DEP *a*)	: *a* を WM の先頭に加える。
(REP #*n a b*)	: #*n* の指す ME 中の要素 *a* を *b* で置換する。
(ERASE #*n a*)	: #*n* の指す ME 中の要素 *a* を消去する。
(APPEND #*n a*)	: #*n* の指す ME の最後の要素として *a* を挿入する。
(LABEL #*n a*)	: #*n* の指す ME の先頭に *a* を挿入する。
(MOVE #*n a*)	: *a* は整数を表わす文字列。#*n* の指す ME を WM 中の *a* 番目の位置に移動する。
(PART *a b*)	: 文字列 *a* を文字に分解し，*b* のラベルを持つ ME として WM に加える。 (例 (PART ABC L1) により (L1 A) (L1 B) (L1 C) が WM に加えられる)
(WORD *a* ?*x*)	: ?*x* は変数。*a* を第1要素とする ME を集め，その第2要素の文字をまとめて1つの文字列とし，?*x* の値とする。 (例 WM が (L1 A) (L1 B) (L1 C) であるとき，(WORD L1 ?*w*) の実行によって "?*w*" に "ABC" が代入される)
(ADD *a b* ?*x*)	: *a*, *b* は整数を表わす文字列。*a*+*b* の値を ?*x* に代入する。
(SUB *a b* ?*x*)	: *a*−*b* の値を ?*x* に代入する。
(MUL *a b* ?*x*)	: *a*×*b* の値を ?*x* に代入する。
(DIV *a b* ?*x*)	: *a* div *b* の値を ?*x* に代入する。
(RETURN)	: 手続きの実行を終了し，その手続きを呼び出した箇所から実行を再開する。
(SAY *a*)	: *a* を出力する。

図8のプログラムは PR の生成と削除を反復して正しく予測を行うための PR を発見する。"FOLLOW" は主プログラムであり，WM を初期値化して "COMPLETE" を呼ぶ。WM にはアルファベットの種類と順序に関する知識として，(ALPH A B……) という ME が含まれることが仮定されている。"COMPLETE" は前述の，予測のためのアルゴリズムに対応した制御を行なう。PR 1によって文字間の関係を求め予測のための PR を生成し，PR 3によって予測を行なう。予測の誤りは PR 2によって検出する。誤りが検出されたときには生成した PR をすべて削除し，仮定した周期を1だけ増加する。最後まで予測を終了すると PR 4によって後続文字列を出力する。"MAKERULE" は与えられた文字列の最初の周期分を取り出し，相対的に同じ位置にある文字間の関係から予測のための PR を生成し "RULES" に加える。文字間の関係は "FINDDIFF" によって求める。"FAILURE" は "RULES" 内に生成された PR をすべて削除する。生成された PR は皆，少なくも1回は実行されるので，WM を調べることによってこれらの PR の RHS を知ることができる。"NEXT" は与えられた文字列の中のある文字に対し，仮定した周期で文字列を区切ったときにその文字が周期の中のどの位置を占めるかを求め，"RULES" を用いて1周期後の文字を予測する。"SETALPH" と "ROTALPH" は2つの文字の間の関係を求めるとき，および，ある文字と，与えられた関係にある文字を求めるときに用いられる。

```
((FOLLOW ?S) = (CLEAR ALPH) (PERCEIVE ?S S)
  (DEP (PERIOD 1) (ORDER 1) (MAKERULE)) (COMPLETE))

(COMPLETE)
(1 (MAKERULE) (PERIOD ?1) => (REM #1) (MAKERULE ?1))
(2 (R ?1 ?2) (S ?1 ?3) (<> ?2 ?3) (PERIOD ?4) => (REMALL R) (FAILURE)
  (ADD ?4 1 ?A) (REP #3 ?4 ?A) (DEP (MAKERULE)))
(3 (ORDER ?1) (S ?1 ?2) (PERIOD ?3) => (NEXT ?1 ?2 ?3) (ADD ?1 1 ?A)
  (REP #1 ?1 ?A))
(4 => (WRITE) (RETURN))

(MAKERULE ?P)
(1 (ORDER ?1) (< ?P ?1) => (REP #1 ?1 1) (RETURN))
(2 (ORDER ?1) (S ?1 ?2) => (ADD ?1 ?P ?A) (FINDDIFF ?1 ?2 ?A)
  (ADD ?1 1 ?A) (REP #1 ?1 ?A))
(3 (ORDER ?1) => (REP #1 ?1 1) (RETURN))

(FINDDIFF ?O ?A ?N)
(1 - (ROTALPH) => (DEP (ROTALPH O)) (SETALPH ?A))
(2 (ALPH ?1) (S ?N ?1) (ROTALPH ?2) => (REM #3) (COND (= ?POS ?O))
  (ACTION (SETALPH ?ORG) (RESP ?2) (RETURN)) (PROD RULES) (RETURN))
(3 (ALPH ?1) (ROTALPH ?2) => (ERASE #1 ?1) (APPEND #1 ?1) (ADD ?2 1 ?A)
  (REP #2 ?2 ?A))

(FAILURE)
(1 (RFOUND) => (REM #1) (REMRULE))
(2 (RESP ?1) => (CHECK RULES (SETALPH ?ORG) (RESP ?1) (RETURN)) (REM #1))
(3 (ORDER ?1) => (REP #1 ?1 1) (RETURN))

((NEXT ?N1 ?N2 ?N3) = (SUB ?N1 1 ?S) (DIV ?S ?N3 ?D) (MUL ?D ?N3 ?M)
  (SUB ?S ?M ?S) (ADD ?S 1 ?A) (RULES ?N2 ?A))

(RULES ?ORG ?POS)

((RESP ?R) = (ROTALPH ?R) (DEP (RESP ?R)))

(SETALPH ?S)
(1 (ALPH ?S) => (RETURN))
(2 (ALPH ?1) => (ERASE #1 ?1) (APPEND #1 ?1))

(ROTALPH ?R)
(1 (= ?R 0) (ALPH ?1) (ORDER ?2) (PERIOD ?3) => (ADD ?2 ?3 ?A)
  (DEP (R ?A ?1)) (RETURN))
(2 (ALPH ?1) => (ERASE #1 ?1) (APPEND #1 ?1) (SUB ?R 1 ?S) (ROTALPH ?S)
  (RETURN))

((PERCEIVE ?S ?L) = (PART ?S ?L) (NUMBER ?L 1))

(NUMBER ?N1 ?N2)
(1 (?N1) => (REP #1 ?N1 ?N2) (LABEL #1 NUMBER) (ADD ?N2 1 ?A)
  (NUMBER ?N1 ?A) (RETURN))
(2 => (REPALL NUMBER ?N1) (RETURN))

(WRITE)
(1 (S ?1) (R ?1) => (REM #1) (REM #2))
(2 (R ?1 ?2) => (ERASE #1 ?1))
(3 => (REVERSE R) (WORD R ?W) (SAY ---> ?W) (RETURN))

(REVERSE ?R)
(1 (?R) => (REP #1 ?R REVERSE) (MOVE #1 1))
(2 => (REPALL REVERSE ?R) (RETURN))

(REPALL ?R1 ?R2)
(1 (?R1) => (REP #1 ?R1 ?R2))
(2 => (RETURN))

(REMALL ?R)
(1 (?R) => (REM #1))
(2 => (RETURN))
```

図8 文字列の予測を行なうプログラム

"PERCEIVE" は文字列を文字に分解し，各文字の文字列内での位置を表わす数とラベルを持つ ME として WM に加える行為である。このとき，文字の位置を求めるために "NUMBER" を用いている。"REVERSE" はその引数を第1要素に持つ ME を見つけ，後にある ME ほど先頭に近い位置に くるように それらを並べ換える。 "REPALL" は第1引数を 最初の要素として 持つ ME をすべて見つけその要素を第2引数で置換する手続き，"REMALL" はその引数を最初の要素とする ME をすべて消去する手続きである。

図9は図8のプログラムの実行例である。 簡単のために アルファベットは A～G の7文字としている。 図の(1), (2) はそれぞれ， 周期1，周期2の仮定に基く PR の生成とその削除を表わし，(3) は周期3における PR の生成を表わしている。 周期3で正しい PR を発見し，後続部分を "EFF" と予測している。

```
< ATTEND >
def(alph a b c d e f g);

< ATTEND >
follow abbcddi

PR    1 WAS CREATED.                                                    }
      (1 (= ?POS 1) => (SETALPH ?ORG) (RESP 1) (RETURN))                } (1)
PR    1 WAS REMOVED.                                                    }
PR    1 WAS CREATED.                                                    }
      (1 (= ?POS 1) => (SETALPH ?ORG) (RESP 1) (RETURN))                }
PR    2 WAS CREATED.                                                    } (2)
      (2 (= ?POS 2) => (SETALPH ?ORG) (RESP 1) (RETURN))                }
PR    2 WAS REMOVED.                                                    }
PR    1 WAS REMOVED.                                                    }
PR    1 WAS CREATED.                                                    }
      (1 (= ?POS 1) => (SETALPH ?ORG) (RESP 2) (RETURN))                }
PR    2 WAS CREATED.                                                    } (3)
      (2 (= ?POS 2) => (SETALPH ?ORG) (RESP 2) (RETURN))                }
PR    3 WAS CREATED.                                                    }
      (3 (= ?POS 3) => (SETALPH ?ORG) (RESP 2) (RETURN))                }
---> EFF
< ATTEND >
list rules;

      (RULES ?ORG ?POS)                                                 }
      (3 (= ?POS 3) => (SETALPH ?ORG) (RESP 2) (RETURN))                }
      (2 (= ?POS 2) => (SETALPH ?ORG) (RESP 2) (RETURN))                } (4)
      (1 (= ?POS 1) => (SETALPH ?ORG) (RESP 2) (RETURN))                }
```

図9 プログラムの実行例

4. APSH の内部仕様

4.1 データの表現

APSH のデータは， 内部ではポインタ，ノード，アトムを構成要素とする木構造によって表現されている。図10はこれらの構成要素を模式的に表わしたものである。ポインタは他のデータを指し示すためのデータである。ノードは木構造の分岐を表現するためのデータで，属性と2つのポインタを持つ。アトムは木構造の末端に接続されるデータで，属性，2つのアトム，印字名（そのアトムに対応して出力される文字列），およびアトムのサイズ（そのアトムが記憶領域上に占める大きさ）からなる。 アトムは辞書式の順序に並ぶように，ポインタ$_2$によって1列に連結されている。属性にはそのアトムが変数を表わすか，整数を表わすか，あるいは行為名を表わ

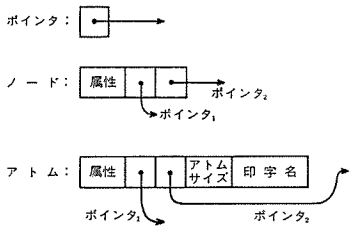

図10 データの構成要素

すか等を識別するための値が書き込まれる。定義行為名または手続き名として使用されているアトムのポインタはその行為の定義内容を表現した木を指す。

4.2 不要なノードの回収

ある木が不要になると，その木を構成していたノードは回収され1列につながれる（図11）。回収されたノードは次に木を生成するときに端から順に取り出され，再利用される。木を作るとき，回収されたノードが残っていなければ新しいノードが作られる。

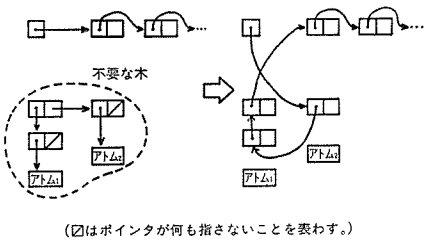

図11 不要なノードの回収

4.3 定義行為，手続きの表現

ユーザが定義する行為や，WM も木によって表現される。図12に，データの外部表現と内部表現との対応の例を示す。図13は定義行為，手続きに対する表現である。

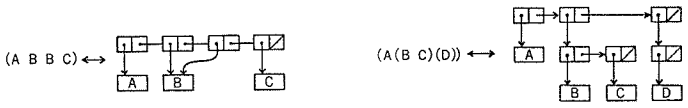

図12 外部表現と内部表現の対応の例

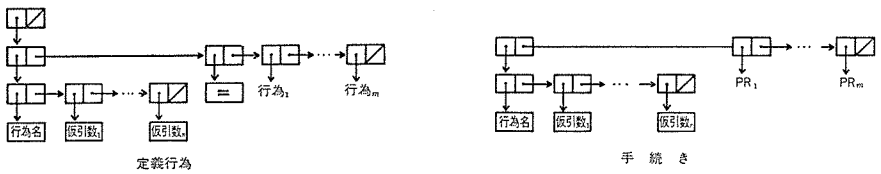

図13 定義行為，手続きの構造

4.4 プログラムの実行制御

APSH の制御は図14のようにスタック状に結合されたノードによって行なわれる。スタックの操作は次のように行なわれる。

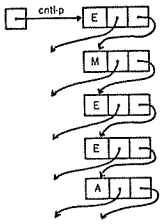

図14 制御スタック

① APSH が起動されたとき，基本行為 "ATTEND" が実行されたとき，またはエラーが発生したときには属性値 "A" を持つノードがスタックの先頭 (cntl-p の指す位置) に加えられる。このノードのポインタ$_1$は入力されたコマンドを表現した木を指す。コマンドの実行が終了するとその木を消去し，次のコマンドを入力してその木を指す。このノードは基本行為 "CON" の実行により除去される。

② 手続きが呼ばれると属性値 "M" のノードがスタックに加えられる。このノードは LHS の満足される PR を選択する状態を表わし，そのポインタ$_1$は手続きの先頭 PR を指す。このノードは手続きの実行を終了する基本行為 "RETURN" の実行により除去される。

③ 定義行為が呼び出されたとき，または手続きの実行において LHS の満足される PR が見つけられたときには，属性値 "E" を持つノードが加えられる。このノードは定義行為または PR の RHS の行為列を実行している状態を表わす。そのポインタ$_1$は行為列の中の，実行中の行為を指す。その実行が終了するとポインタ$_1$の指す位置は右隣りの行為に移動する。行為列の最後まで実行が終了すると，このノードは除去される。

4.5 変数の管理

変数とその変数に代入されている値の対応関係は，ポインタ$_1$が変数を指し，ポインタ$_2$が値を指すノードによって表わす。この変数値対は図15 (a) のように1列につながれており，新たに変数に値が代入されると，それに対応した対が先頭 (var-p の指す位置) に加えられる。

APSH では定義行為，手続き内で値を参照できる変数はその行為内で値が代入されたもののみに限っている。この制限は図15 (b) のような「参照禁止ポインタ」で表わしている。参照禁止ポインタは定義行為，手続きの呼び出しに対応して生成され，スタックを形成する。

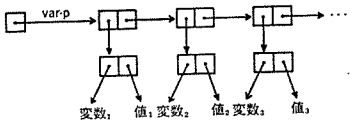

図15 (a) 変数と値の対応の表現

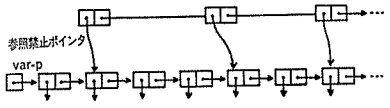

図15 (b) 参照禁止ポインタ

5. あとがき

適応プロダクションシステム APSH の機能とプログラムの例，および内部仕様について述べた。APSH は次の点を考慮して作成された。

① プログラムによって PR の生成，削除，参照を行なうための機能を取り入れ，PR のデータ的性格を強める。

② 階層的な PM や逐次的に実行される PM の例のような構成を可能にし，制御構造の複雑

なプログラムを記述できるようにする。

現在，本論文で扱った例題のほかにもいくつかのプログラムを APSH 上で作成し，実行している。その結果，上記の点について，期待された効果が得られたように思う。しかし，今後の問題点として次のようなことを検討して行く必要性も認められた。

① 条件判断による分岐はすべて PR の選択という形式に依存しているため，PR が多用され，プログラムが繁雑になる傾向がある。

② PR の RHS が複雑になると，操作を加える PR の指定のためにその完全な形を与えることは困難になると考えられる。

③ 現在のシステムでは操作を加えられる PR は暗黙のうちに1つに仮定されているが，PR の一般化のために PR を比較する場合などは，同時に複数個の PR を指定できることが望ましい。

以上のような問題点に対し，たとえば①では PR の RHS に IF-THEN-ELSE の構造を取り入れる，③では PR を指すための変数を導入する，などの対策が考えられるが，これらの機能の拡張は PS 本来の特長を損う場合もあると思われる。今後は認知科学，知識工学の分野における具体的な問題に APSH を適用することによってシステムの能力を試験するとともに，改善すべき点，改善の方法について検討して行く予定である。

APSH は北大大型計算機センターの HITAC M-200 H VOS 3 上で実現されている。プログラムは FORTRAN 77 で書かれており，現在のステップ数は約 2,800 である。

本研究を進めるにあたり，御指導，御助言を頂いた工学研究科情報工学専攻の竹村伸一教授，APSH の機能決定に関して御討論頂いた文学部行動科学科の阿部純一講師に感謝の意を表します。

参 考 文 献

1) 辻井潤一：情報処理，20, 8, p. 735～743 (1979)

2) Waterman, D. A.: Adaptive Production Systems, Proc. of IJCAI-75, p. 296～303 (1975)

3) 安西祐一郎：計測と制御，18, 4, p. 303～311 (昭54)