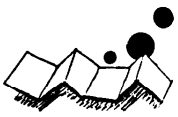
解　説



# プロダクションシステム†

小　林　重　信††

## 1. はじめに

プロダクションシステム（以下PSと略す）はPost機械にその理論的基礎をおく計算モデルの1つである．Turing機械に基礎をおく従来のプログラムが制御駆動的であるのに対し，PSは事象駆動的であることを大きな特徴とする．事象駆動的表現は悪構造問題のモデル化に有用な枠組みを提供する．悪構造問題では，そこで使われる知識の範囲，知識間の相互関連をあらかじめ大局的に把握することが困難であり，if-then形式の表現が適している．これまで多くのエキスパートシステムがPSを用いて開発されてきた理由はここにある．

本解説では，まずPSの基本モデルを概説する．つぎにpure PSの枠組みのもとで開発されている代表的なPS記述言語をいくつか紹介する．ついでpure PSの本質的問題を議論，協調的問題解決の必要性を指摘，いくつかのモデルを紹介する．さらにPSの応用と特徴をまとめ，今後の課題を考察する．

## 2. プロダクションシステムの構造と動作[1),2)]

### 2.1 PSの構造と認識行動サイクル

PSの構造は図-1に示される．プロダクションメモリ（PM）にはルールと呼ばれる“前提→結論”の対，ワーキングメモリ（WM）にはルールによって参照・更新の対象とされるデータがそれぞれ格納される．インタプリタはPMとWMの両方を参照，実行可能なルールを見い出し，その中の1つを実行に移す．

実行はつぎの認識行動サイクルの繰り返しである．

1) 照合（matching）：WMとPMの間で照合を行い，条件を満たすすべてのルール（競合集合）を求める．

2) 競合解消（conflict resolution）：競合集合の中から選択基準に従って，特定のルールを選択する．

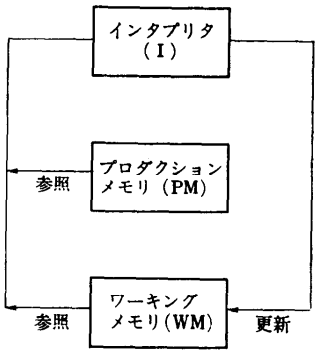



図-1　プロダクションシステムの構造

3) 実行（action）：2)で選択されたルールの行動部の動作を実行し，WMの内容を更新する．

### 2.2 推論の方向

推論の方向について，特徴と応用例を示す．

**1) データ駆動型推論**（data driven reasoning）

あらかじめ観測されたデータをWMに与えておき，ルールの前提部との照合を行い，結論部で指示される内容をWMに追加することを繰り返して推論を進める方式をいう．有機化合物の構造同定システムDENDRAL[3)]や計算機構成システムR1[4)]などがこの方式を採用している．

**2) 事象駆動型推論**（event driven reasoning）

1)の特殊な場合で，データが時間の結過につれて入力されるとき，もっとも最近に入力されたデータまたは推論結果に反応して前向きに推論を進める方式をいう．集中医療監視システムVM[5)]や計算機オペレーション支援システムYES/MVS[6)]などがこの方式を採用している．

**3) 目標駆動型推論**（goal driven reasoning）

目標が初めに与えられており，これを証明するためにルールを後向きに適用する方式をいう．感染症診断システムMYCIN[7)]や計画作成システムABSTRIPS[8)]などがこの方式を採用している．

† Production System by Shigenobu KOBAYASHI (Tokyo Institute of Technology Depatment of Systems Science).

†† 東京工業大学総合理工学研究科

### 2.3 競合解消戦略

現在の WM に対し適用可能なルールが複数存在するとき競合解消のためにつぎのような戦略が使われる.

1) 重要度優先方式：重要度の高い順に照合を行い，照合すればそのルールを選択する.
2) 詳細優先方式：条件部の記述がもっとも詳細なルールを優先して選択する.
3) 最近規則優先方式：もっとも最近に実行されたルールを優先して選択する.
4) 最近 WM 要素優先方式：もっとも最近に WM に追加された要素に照合するルールを優先して選択する.
5) 並列実行方式：照合するルールをすべて実行する.

## 3. OPS5 による知識の表現と利用

現在もっとも広く使われている PS 記述言語として OPS5[9]がある．ここでは OPS5 の概要を例題とともに示す.

### 3.1 OPS5 のデータ構造

WM 要素はつぎのように定義される.

〈WM 要素〉::=(〈対象名〉〈属性名〉〈値〉
・・・ 〈属性名〉〈値〉)

例："円板1が柱Bにある"
(disk ^size 1 ^place B)

ここで "^" は属性名を表す識別子である.

ルールはつぎの形式で表現される.

(*p* 〈条件部〉 — —〉〈行動部〉)

条件部は条件要素の連接である．条件要素はパターン記述であり，変数の使用が許される.

例："任意の柱にある大きさ2の円板"
(disk ^size 2 ^place 〈*x*〉)

ここで 〈*x*〉 は変数を表す．否定を表す条件要素はパターンの前に "−" をつけて区別される.

属性の値の範囲を規定するために述語が使われる.

例："大きさが3以上の任意の柱にある円板"
(disk ^size 〉 3 ^place 〈*x*〉)

"〉" 以外に，"=", "〈〉" (等しくない)，"〈=〉" (同じ型) などの述語が用意されている．また値の範囲を and 条件および or 条件で記述するために，それぞれ "{ }" および "《 》" が使われる.

例："大きさが 3〜7，AまたはBの柱にある円板"
(disk ^size {〉= 3 =〈 7} ^place 《A B》)

ルールの行動部には，WM を更新するための操作，計算の実行，手続きの呼出しなどが記述される．WM の更新操作としては，make, remove, modify がある．属性の値を計算式で求めたり，手続きを呼び出すために，(compute 〈計算式〉) や (call 〈手続き名〉) が使われる.

### 3.2 OPS5 の競合解消戦略

OPS5 では WM は，〈タイムタグ，WM 要素〉なる順序対の集合である．タイムタグは WM 要素の生成順序を表わす通し番号である．発火可能なルール名とこれに照合する WM 要素のタイムタグの対をインスタンシエイションと呼ぶ．2つの競合解消戦略がある．LEX 戦略ではインスタンシエイションの肯定パターンに対応するタイムタグを番号の大きい順に並べ，辞書式順番にしたがって選択する．また MEA 戦略では条件部の1つ目の肯定パターンに照合する WM 要素のタイムタグが最大のインスタンシエイションを選択する．同じ場合には，以下に LEX 戦略を適用する.

### 3.3 例題（ハノイの塔）

柱A，B，Cがあり，最初Aの柱に大きさの異なる $N$ 個の円板が上から小さい順に入っている．小さい円板の上に大きい円板をのせられないことおよび1度に1つの円板しか動かせないという制約のもとに，Aの柱のすべての円板をCの柱に移すことが問題である.

この問題は OPS5 でつぎのようにモデル化される.

① move_disk：大きさ 〈*p*〉 の円板を場所 〈*x*〉に移すことが目標であるとき，〈*p*〉 の円板が場所 〈*y*〉 にあり，場所 〈*x*〉，〈*y*〉 のどちらにも 〈*p*〉 より小さい円板が存在しないならば，円板 〈*p*〉 を場所 〈*x*〉 に移す．そしてC以外の場所にあって，〈*p*〉 を除いて最大の円板の大きさ 〈*q*〉 を求め，これを場所Cに移すことをつぎの目標に設定する.

② move_last_disk：②の特殊な場合で，最後の1枚を移すときにはつぎの目標を設定する必要がない.

③ change_subgoal：大きさ 〈*p*〉 の円板を場所 〈*x*〉 に移すことが目標であるとき，〈*p*〉 の円板が場所 〈*y*〉 にあり，場所 〈*x*〉，〈*y*〉 のいずれかに 〈*p*〉 より小さい円板が存在するならば，この目標を取消し，〈*x*〉 または 〈*y*〉 にあって，〈*p*〉 より小さい円板の中で最大のものを 〈*q*〉 とし，円板 〈*q*〉 を 〈*x*〉，〈*y*〉 以外の第3の場所 〈*z*〉 に移すことをつぎの目標として設定する.

**図-2，図-3** に例題の OPS5 による表現と実行が示

される.

### 3.4 OPS5 の特徴

OPS5 は，前向き，後戻りなしのデータ駆動型推論方式である．WM はグローバルで均質な構造である．他の PS と比べてプログラミング言語としての汎用性および完成度が高い．ただしプログラミング環境はあまり用意されていない．OPS5 は RETE match と呼ばれる高速の照合メカニズムをもつ．なお推論の高速化については本誌 60 年 3 月号の石田の解説[10]および計測と制御 60 年 8 月号小林の解説[11]を参照されたい．

```
1: ; Tower of Hanoi
2:
3: (literalize goal size place)
4: (literalize disk size place)
5: (literalize pole place)
6:
7: (p move_disk
8:    (goal ^size ⟨p⟩ ^place ⟨x⟩)
9:    (disk ^size ⟨p⟩ ^place ⟨y⟩)
10:  -(disk ^size ⟨⟨p⟩ ^place ⟨⟨x⟩ ⟨y⟩⟩⟩)
11.   (disk ^size {⟨q⟩ ⟨⟩ ⟨p⟩} ^place ⟨⟩ c)
12:  -(disk ^size {⟩ ⟨q⟩ ⟨⟩ ⟨p⟩} ^place ⟨⟩ c)
13: --⟩
14:   (modify 2 ^place ⟨x⟩)
15:   (remove 1)
16:   (make goal ^size ⟨q⟩ ^place c))
17:
18: (p move_last_disk
19:   (goal ^size ⟨p⟩ ^place ⟨x⟩)
20:   (disk ^size ⟨p⟩ ^place ⟨y⟩)
21:  -(disk ^size ⟨⟨p⟩ ^place ⟨⟨x⟩ ⟨y⟩⟩⟩)
22:  -(disk ^size ⟨⟩ ⟨p⟩ ^place ⟨⟩ ⟨x⟩)
23: --⟩
24:   (modify 2 ^place ⟨x⟩)
25:   (remove 1))
26:
27: (p change_goal
28:   (goal ^size ⟨p⟩ ^place ⟨x⟩)
29:   (disk ^size ⟨p⟩ ^place ⟨y⟩)
30:   (disk ^size{⟨q⟩ ⟨ ⟨p⟩} ^place⟨⟨x⟩ ⟨y⟩⟩⟩)
31:  -(disk ^size{⟩ ⟨q⟩ ⟨ ⟨p⟩} ^place ⟨⟨x⟩ ⟨y⟩⟩⟩)
32:   (pole ^place{⟨z⟩ ⟨⟩ ⟨x⟩ ⟨⟩ ⟨y⟩})
33: --⟩
34:   (remove 1)
35:   (make goal ^size ⟨q⟩ ^place ⟨z⟩))
36:
37: ; initial state for 3 disks·····
38:
39: (make pole ^place a)
40: (make pole ^place b)
41: (make pole ^place c)
42: (make disk ^size 3 ^place a)
43: (make disk ^size 2 ^place a)
44: (make disk ^size 1 ^place a)
45: (make goal ^size 3 ^place c)
```

図-2 OPS 5 によるハノイの搭のモデル化

```
*(wm)
   1: pole ^place a   2: pole ^place b
   3: pole ^place c
   4: disk ^size 3 ^place a      5: disk ^size 2 ^place a
   6: disk ^size 1 ^place a      7: goal ^size 3 ^place c
*(run)
  1)apply⟨change_goal, (7, 4, 5, 2)⟩.      remove 7.
    8: goal ^size 2 ^place b
  2)apply⟨change_goal, (8, 5, 6, 3)⟩.      remove 8.
    9: goal ^size 1 ^place c
  3)apply⟨move_disk,, (9, 6, 4)⟩.          remove 6, 9.
   10: disk ^size 1 ^place c     11: goal ^size 3 ^place c
  4)apply⟨change_goal, (11, 4, 5, 2)⟩.     remove 11.
   12: goal ^size 2 ^place b
  5)apply⟨move_disk, (12, 5, 4)⟩.          remove 5, 12.
   13: disk ^size 2 ^place b     14: goal ^size 3 ^place c
  6)apply⟨change_goal, (14, 4, 10, 2)⟩.    remove 14.
   15: goal ^size 1 ^place b
  7)apply⟨move_disk, (15, 10, 4)⟩.         remove 10, 15.
   16: disk ^size 1 ^place b     17: goal ^size 3 ^place c
  8)apply⟨move_disk, (17, 4, 13)⟩.         remove 4, 17.
   18: disk ^size 3 ^place c     19: goal ^size 2 ^place c
  9)apply⟨change_goal, (19, 13, 16, 1)⟩.   remove 19.
   20: goal ^size 1 ^place a
 10)apply⟨move_disk, (20, 16, 13)⟩.        remove 16, 20.
   21: disk ^size 1 ^place a     22: goal ^size 2 ^place c
 11)apply⟨move_disk, (22, 13, 21)⟩.        remove 13, 22.
   23: disk ^size 2 ^place c     24: goal ^size 1 ^place c
 12)apply⟨move_last_disk, (24, 21)⟩.       remove 21, 24.
   25: disk ^size 1 ^place c
 )There is no instantiations.
*(wm)
1: pole ^place a    2: pole ^place b
3: pole ^place c
18: disk ^size 3 ^place c          23: disk ^size 2 ^place c
25: disk ^size 1 ^place c
```

図-3 OPS 5 によるハノイの搭の実行

## 4. 他の代表的なプロダクションシステム言語

### 4.1 診断向きプロダクションシステム

医療診断や地質学データの解釈を対象として開発されたエキスパートシステムのデータ構造や推論エンジンをツール化することにより，主として診断分野に適した PS がつくられている．

#### 4.1.1 EMYCIN[12]

EMYCIN はスタフォード大学で開発された感染症診断システム MYCIN を起源とする PS である．

EMYCIN が扱う基本データ構造はつぎの 3 項組である．

〈3 項組〉::=(〈コンテキスト〉〈属性〉〈値〉)

ここでコンテキストは患者，検体，細菌などの対象を表すパラメータであり，これらはコンテキストツリー

と呼ばれる階層を構成する．各コンテキストは複数の属性により特徴づけられる．各属性はその型，取り得る範囲および値の決定方法などが事前に定義される．

すべての3項組には確実度と呼ばれる尺度値が割り当てられる．ルールの条件部は条件要素の積和標準形で表現される．条件要素はつぎの形式である．

〈条件要素〉::=(〈述語〉〈3項組〉)

ここで述語は，〈3項組〉の不確実性を評価し，真または偽を与えるもので，SAME (…らしい)，MIGHTBE (…かもしれない) などが用意されている．

ルールの結論部はつぎの形式である．

〈結論部〉::=(〈3項組〉〈確実度〉|〈手続き〉)

EMYCIN では後向き連鎖にしたがい推論が進められる．EMYCIN には，ユーザやシステム構築者に対し推論の過程を説明する WHY 機能や HOW 機能，内部表現を自然言語風の外部表現に変換するトランスレータ，ルール間の矛盾や包含関係をチェックするルールベースエディタ，デバッガなど大規模なシステム開発にとって便利なユーティリティが豊富に用意されている．

EMYCIN の欠点としては，不確実性を伴う後向き推論であるためにすべての目標を調べることが必要であり，コンサルテーションに多くの時間がかかり，現場向きでないこと，データ駆動的な処理が十分に行えないこと，時間の経過を陽に扱えないことなどが指摘される．

#### 4.1.2 KAS[13]

KAS は SRI インタナショナルで開発された鉱物資源探査システム PROSPECTOR を起源とする PS である．

KAS が扱う基本データ構造はつぎの2項関係である．

〈ステートメント〉::=(〈関係〉〈対象1〉〈対象2〉)

各対象は分類階層木の1つの節点に対応づけられている．ルールの条件部は複数のステートメントの論理的結合で表現される．結論部はつぎのように表される．

〈結論部〉::=(〈重み LS〉〈重み LN〉
〈ステートメント〉)

ここで LS の値は証拠の仮説に対する十分性を表す重みで，条件部が成立するとき結論部のステートメントの確実度を高めるのに使われる．一方 LN の値は証拠の仮説に対する必要性を表す重みで，条件部が成立しないとき，結論部のステートメントの確実度を低めるのに使われる．確実度の算出法は主観的ベイズ法による．

KAS の推論方法は EMYCIN のように固定されたものではなく，相互主導型 (mixed initiative) である．すなわちシステム側は目標仮説の選択とそれを確かめるためのユーザへの質問を考慮するのに対し，ユーザは自分にとって好ましい目標仮説の指示や事実の入力を行うことができる．目標仮説を構成するステートメントの中から影響度の大きいものを選択し，ユーザに質問を発し，応答結果を前向き推論によって関係する他のステートメントに伝播した後つぎのゴールを選択することを繰り返す．またゴール仮説の選択を制御するために，コンテキストと呼ばれる関係が使われる．

KAS が扱うデータ構造は2項関係，すなわち意味ネットワークであり，ネットワークを直接扱うためのエディタが用意されている．また知識ベースの保守を容易にするために，知識の追加に対し感度テストが行えるなどの利点がある．KAS は2項関係および分類学的な知識を前提としており，より一般的な知識を陽に扱う枠組をもたないため，汎用性に欠ける．

### 4.2 汎用プロダクションシステム言語

2章で取り上げた OPS5 の他に，汎用 PS 言語としては ROSIE, OPS83 などがある．

#### 4.2.1 ROSIE[14]

ROSIE は RAND 社で開発された汎用の PS 言語である．ROSIE では，知識は宣言的な事実が格納されるデータベースと，データベースを操作するための複数の規則集合（プログラム）に分けて表現される．宣言的知識は legal relational forms と呼ばれる制限された英文によって表現されるが，内部的には $n$ 項関係に変換される．“each of”, “every”, “any” などのクラスの要素の扱いも可能である．また “which is”, “who does” などの関係詞節の表現も許される．

手続き的な知識は規則集合として表現される．規則集合は複数存在し得る．規則はデータベースに対する登録・検索・更新および表示などの操作を指示することを基本とするが，loop や if-then-else を始め，一般のプログラミング言語にあるより複雑な手続きを記述することもできる．

特定の規則集合を活性化する命令を送ることにより，その規則集合での結論が開始される．各規則集合内では，推論はつぎの3通りのモードで進められる．逐次実行では，集合内の規則が上から順に実行される．循環実行では “return” 動作に到達するまで逐次

実行が繰り返される．ランダム実行では規則がランダムに選択され，実行される．

ROSIE は英語的な知識の表現と利用を許すことにより，可読性の高い知識ベースの構築を可能にしている．推論方法は固定されていないために，通常の PS に比べてよりプログラミング言語的性格が強いといえる．ROSIE は DEC-20 の INTERLISP 上に実現されているが，システム効率やメモリ制約のために，実験的システムの構築がなされているに過ぎないが，その野心的な試みは評価される．

#### 4.2.2 OPS83[15]

OPS83 は OPS5 に続くつぎのバージョンとしてカーネギーメロン大学で開発された汎用 PS 言語であるが，OPS5 と比べて大幅な仕様の変更が行われている．ここでは OPS5 との違いを中心に述べる．

OPS5 が人工知能や知識工学の基礎的研究のツールであったのに対し，OPS83 はエキスパートシステム開発ツールを意図している．OPS83 は ADA, C, PASCAL などがもつ通常のプログラミングパラダイム（データタイプ，実行ステートメント，モジュール，入出力など）を基本的にすべてサポートしており，PS は OPS83 の中では1つの構成要素として位置づけられている．

WM 要素については，属性値として各種のデータ型をあらかじめ宣言することによって利用することができる．属性値を規定するために，OPS5 における算術式のほかに，あらかじめ定義された手続きのルーチンコールが利用できる．

ルールの右辺は OPS5 におけるパターン書き換え操作のほかに，通常のプログラミング言語における手続きを記述できる．ルールの左辺にはパターン記述が書かれるが，パターンの中での関数呼出しが許される．

OPS5 では LEX および MEA という2つの競合解消戦略しか用意されていないが，OPS83 では競合解消はユーザに完全に開放されている．ユーザは解消戦略を手続きを用いて記述できる．また WM を調べる関数や実行をトレースする関数がいくつか利用可能である．

要するに OPS5 が pure PS の枠組の中で汎用性を追求したのに対し，OPS83 は PS を既存のプログラミングパラダイムの中に埋込むことにより，実用システムの開発を促進することを追求しているといえる．

## 5. PS における協調型問題解決

### 5.1 協調型問題解決の必要性

PS を用いて，大規模複雑なシステムをモデル化する際の1番大きな障害は，すべての状況に対応できる単純で強力な競合解消戦略を設計することが困難なことにある．これは制御の飽和問題と呼ばれるもので，現実的に解決するには，制御に関する知識をルールとして記述し，対象に関する知識とともに，PM に登録しておくことが必要となる[16]．しかしこれは知識と制御を分離して表現する PS の基本思想に反し，知識ベースの管理を損なう恐れが生じる．

制御の飽和問題を克服する上で，協調型問題解決の枠組が有効である．協調型問題解決とは，知識源 (knowledge sources) が分散され，それらが疎に結合された集まりによる協調的な問題解決をいう[17]．知識源とは，手続きやルールなどの集合で，独立した処理の単位をいう．どの知識源も問題全体を解決する上で十分な情報をもたないので，情報を相互に共有または伝達し合うことにより，問題解決に向けた協調が要請される．

協調型問題解決は，つぎのような利点をもつ．

① 不確かな知識やデータの誤りを吸収できるので高い信頼性が得られる．
② モジュラリティが高く，概念的に明快なので，拡張性が高い．
③ 本来分散的性質をもつ問題を自然にモデル化できる．
④ ハードウェア化により，高速処理が期待できる．

### 5.2 黒板モデル[18]

このモデルは音声理解システム HEARSAY-II[19] で提案されたもので，黒板 (blackboad) と呼ばれるグローバルな階層的データ構造に対し，複数の知識源がその特定の階層にアクセスして，全体として信頼性の高い結論を引き出すことを意図したシステムである．**図-4** に HEARSAY-II における黒板の階層と知識源の関係を示す．**図-5** にその制御構造を示す．知識源は黒板事象と呼ばれる黒板上の特定の変化に反応して活性化される．黒板モニタは活性化された知識源の詳細な条件をチェックし，もしも実行可能であれば，その知識源の実行のインスタンシエーションをつくり，これをスケジューリング行列に送る．スケジューラは現在の制御の焦点 (focus of control) の下で最適なイ

ンスタンシエーションを選択し，これを実行に移す．図-5 の制御方式は事象駆動型である．事象駆動型制御の欠点は誤った情報の混入や必要な情報の欠落に対し弱いことにある．

Corkill[20] らは事象駆動型制御の問題点を克服するために，黒板事象から目標を生成するメカニズムを導入することによって，事象駆動型制御と目標駆動型制御を統合化する方法を提案している．図-6 は拡張モデルの構成を示す．元の(データ) 黒板に加えて，目標黒板が追加されている．新しい制御要素としてプランナが導入されている．プランナは目標駆動型の知識源を選択したり，目標 / 副目標の関係を利用して計画を生成するために利用される．Corkill らはある地域上を動く乗物の動的マップを生成するシステムへの応用を通じて統合化戦略の有効性を確認している．

黒板モデルは，蛋白質結晶構造解析システム CRYSALIS[21]，雑音の混じった信号の解釈システム SV/X と SV/P[22),23)]，マルチバンド航空写真の解釈システムなどに応用されている．また黒板モデルを取り入れたツールとしては，HEARSAY-Ⅲ や AGE[24)]がある．

黒板モデルに類似した分散型 PS としては，オフィス業務の自動化を指向した APN モデル[25]，計算機オペレーションの実時間制御を支援する YES/MVS モデル[6]，事象駆動型システムの表現と制御を指向する Co-PSs[26] などがある．これらに共通する考え方は対象間の相互作用をデータ送信およびメモリ共有によって実現している点にある．

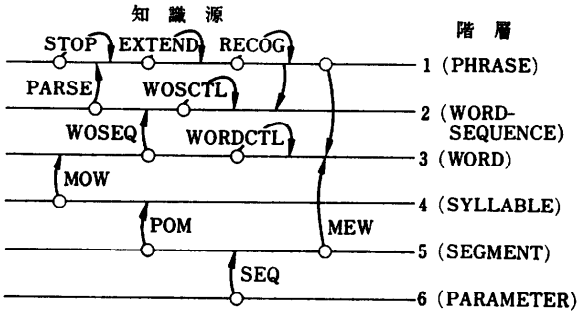
### 5.3 通信/同期機構をもつ分散型 PS

著者らは対象間の相互作用を最近の手続き言語における並列処理や通信 / 同期の機構で表現することに基礎をおく分散協調型 PS 記述言語 Po-PS (Process Oriented Production Systems)[27] を設計・開発し，種々の応用を試みているので紹介する．

Po-PS はシステムを互いに非同期，同時進行な逐次型タスクの集まりで表現する．タスクはつぎのようなメッセージを介した通信 / 同期機構をもつ．

1) 非同期型送信：他のタスクにメッセージを送信する．

2) ブロードキャスト：チャネルに接続されているすべてのタスクにメッセージを送信する．




PRDL：全階層にアクセス

図-4 HEARSAY-Ⅱ におけるブラックボード

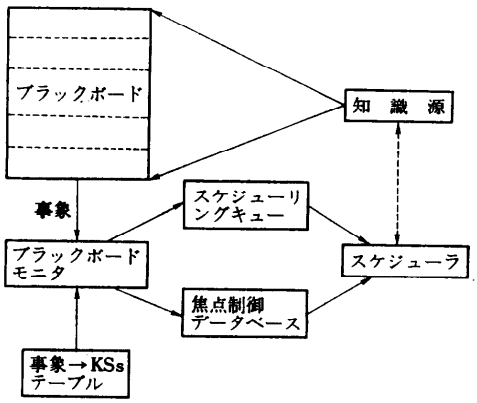



図-5 事象駆動型ブラックボードモデル

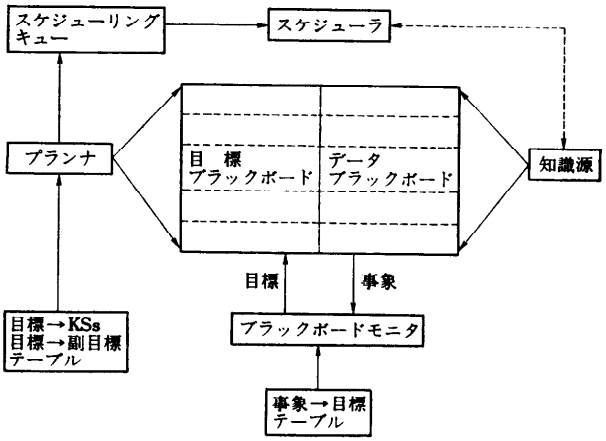



図-6 目標駆動型ブラックボードモデル

3) 同期型送信：他のタスクにメッセージを送り，そのタスクがメッセージを受理するまで待つ．

4) 並列 and (or) 型同期型送信：複数のタスクにメッセージを送信し，それらがすべて（それらの1つが）受理されるまで待つ．

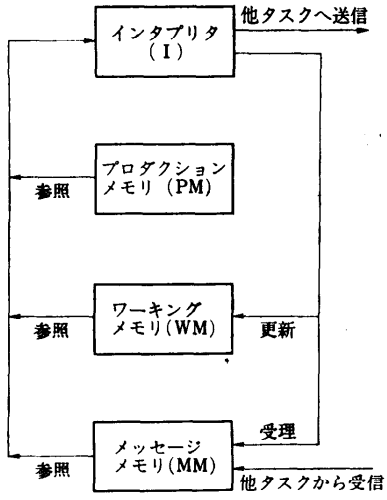



図-7 Po-PS におけるタスクの構造

タスクは通常の PS に上記の通信 / 同期機構を付加した拡張 PS として表現される．タスクの基本構造を図-7 に示す．ここでメッセージメモリ(MM)は他のタスクから送信されたメッセージの記憶場所である．ルールは WM と MM の両方に事象駆動的に反応する．ルールの構文は OPS5 に準拠し，いくつかの拡張がなされている．

タスクはそのプロトタイプであるタスク型の複製として生成される．タスク型とタスクの関係は，対象指向言語におけるクラスと対象の関係に相当する．上位タスク型は，上位クラスに相当する概念であり，各タスク型は，その上位タスク型のパラメータおよび諸宣言を深さ優先で多重継承する．

各タスクのインタプリタは OPS5 の LEX 戦略に基づく認識行動サイクルを繰り返し実行する．Po-PS の処理系は逐次型マシンの上に実現されているために，各タスクは並列に実行されるのではなく，タスクの生成関係を表す階層木上の節を広がり優先で訪れ，タスク型で宣言されてヒット戦略にしたがって実行される．

Po-PS は MELCOM-COSMO 700Ⅲ の LISP 1.9 上に実現されており，種々の同

```
(tasktype scheduler
  hit      : exhaustive
  message  : use      ^no
             release ^no
             remind
  element  : fork     ^no ^status
             reminding ^philosopher
  scheduling :
    s1 (: from ?phil release ^no ?n)
       (fork ^no          ?n      ^status up)
       (fork ^no (+(: mod ?n 5) 1) ^status up)
    =) (modify 2 ^status down) (modify 3 ^status down)
       (accept 1)
    s2 (: from ?phil use)
       (: from ?phil remind)
    -  (: from ?any release)
    =) (accept 2)
       (make reminding ^philosopher ?phil)
    s3 (: from ?phil use ^no ?n)
       (reminding ^philosopher ?phil)
       (fork ^no          ?n      ^status down)
       (fork ^no (+(: mod ?n 5) 1) ^status down)
    -  (: from ?any release)
    =) (modify 3 ^status up) (modify 4 ^status up)
       (accept 1) (remove 2)
    s4 (: from ?phil use ^no ?n)
       (fork ^no          ?n      ^status down)
       (fork ^no (+(: mod ?n 5) 1) ^status down)
    -  (: from ?any release)
    -  (reminding ^philosopher)
    =) (modify 2 ^status up) (modify 3 ^status up)
       (accept 1)
  initial :
  (make fork ^no 1 ^status down)
  (make fork ^no 2 ^status down)
  (make fork ^no 3 ^status down)
  (make fork ^no 4 ^status down)
  (make fork ^no 5 ^status down)
  (create ph-1 philosopher ^no 10 ^dine 50 ^think 40 ^endure 30)
  (create ph-2 philosopher ^no 20 ^dine 10 ^think 50 ^endure 40)
  (create ph-3 philosopher ^no 30 ^dine 20 ^think 10 ^endure 50)
  (create ph-4 philosopher ^no 40 ^dine 30 ^think 20 ^endure 10)
  (create ph-5 philosopher ^no 50 ^dine 40 ^think 30 ^endure 20)
```

(a) Task Type scheduler.

```
(tasktype philosopher ^no ^dinet ^think ^endure
  hit:      single
  element : self ^activity
  duty :
    d1 (self ^activity waiting)
    =) (request (: parent) use ^no *no
          : within *endure
          : else (send (: parent) remind))
       (modify 1 ^activity dining)
    d2 d(self ^ activity dining)
    =) (delay *dine)
       (send (: parent) release ^no *no)
       (modify 1 ^activity thinking)
    d3(self ^activity thinking)
    =) (delay *think)
       (modify 1 ^activity waiting)
  initial : (make self ^activity waiting))
```

(b) Task Type philosopher.

図-8 Po-PS による哲学者の食事問題のモデル化

期問題，雑誌編集過程のモデリング，サービス業務のスケジューリング，線画のラベリングなどに適用されている．図-8 に哲学者の食事問題を変形したものの Po-PS によるモデル化を示す．

Po-PS は従来のモデルに比べ事象駆動型システムのモデル化がより自然に行えることが確認されている．Po-PS は黒板モデルや契約モデルなどの協調型問題解決のパラダイムを陽に組み込んではいないが，これらは Po-PS 上に容易に実現することができる．

## 6. プロダクションシステムの応用と特徴

PS の応用範囲はデータの解釈・診断・監視と制御・計画と設計などあらゆる領域に及んでいる．

### 6.1 PS の応用

データ解釈の分野では，有機化合物の構造同定システム DENDRAL[3]，蛋白質結晶の構造同定システム CRYSALIS[21]，音声理解システム HEARSAY-Ⅱ[19] などがある．データ解釈にはノイズやデータの欠落の処理が不可欠であり，単純な推論戦略ではうまくいかない場合が多い．実際に CRYSALIS や HEARSAY-Ⅱは黒板モデルを採用して，信頼性の高い推論結果を得ている．

診断の分野では，感染症診断システム MYCIN[7]，計算機ハードウェアの故障診断システム SPEAR[25]，電話回線の保守システム ACE[29]を始め，PS の応用がもっとも多くなされている．診断システムでは，不確実性の存在が前提とされる場合が多く，推論の効率と結果の信頼性の間にトレードオフが存在する．また大規模人工システムの場合，設計知識と潜在的故障との関連性を PS 上でどのように表現するかが課題とされる．

監視と制御の分野では，集中医療監視システム VM[5]，計算機オペレーションの監視システム YES/MVS[6]，原子炉の監視システム REACTOR[30] などがあり，我が国でもシステム制御への応用が始まっている[31]．監視・制御システムの特徴は厳しい実時間性が要求されることにある．ルールベースの構造化，高速の照合アルゴリズムの採用が不可欠である．実システムに組み込まれて使われるので，既存システムとの接続性も重要なことである．

計画と設計の分野では，計算機機器構成システム R1/XCON[4]，VLSI 自動設計システム DAA[32]，遺伝子組み替え実験計画システム MOLGEN[33),34] などがある．計画・設計問題は，基本的に組合わせ的問題であるため，組合わせ的爆発を回避するために，抽象的階層化などの探索・計画に関する戦略が必要であり，これを PS の枠組の中でどう実現していくかがモデリングにおける課題となる．

### 6.2 PS の特徴

PS はつぎのような長所をもつ．

1) 思考過程との親和性：if-then 形式の知識表現は，人間の断片的な知識を表現する上で，なじみがよい．
2) モジュール性：ルールの構造的および機能的モジュール性は，知識ベースの更新や拡張を容易にする．
3) 説明可能性：推論の筋道を辿ることが容易であり，説明機能を付加することにより，デバッグやユーザへの理解を向上させることができる．

一方，PS はつぎのような短所をもつ．

1) 副作用の生起：ルールは WM を介して間接的に相互作用を及ぼし合うので，大規模システムでは予期せぬ相互作用が生じて，誤った挙動を示す可能性が高い．
2) 組合わせ的爆発：認識行動サイクルにおける計算時間の大部分はパターンマッチングに費やされるので，組合せ的爆発の問題を引き起こす．
3) 制御の飽和問題：単純な競合解消戦略では，大規模システムに対処できず，制御知識をルールとして表現することが必要になる．

要するに，pure PS の枠組内で大規模システムをモデル化しようとすると，ここで述べた短所が一挙に表面化してくるところに，PS の本質的な問題がある．知識の構造的表現が要請されるゆえんである．

## 7. おわりに

PS はその事象駆動的オペレーションを最大の武器として，これまで悪構造問題をモデル化する上で数多くの成果を上げてきた．同時に PS の潜在的な短所を露呈してきたのも事実である．pure PS として OPS5 はもっとも洗練されたデータ構造や高速照合機能をもち，実用に耐える PS 記述言語である．しかし大規模システムを対象に，可読性や拡張可能性を損なうことなく，OPS5 で自然にモデル化することは非常に困難である．

1つの PS によるモデル化の適性規模はルール数で10～20 程度と思われる．分散協調型 PS は pure PS の短所を克服する上で有用な枠組みと考えられる．

より一般的な知識の表現と利用の観点からみれば，PS は1つのパラダイムに過ぎず，フレームなど他の知識表現形式と合わせて利用することが実際には有効である．LOOPS[35] や KEE などはそのような統合的環境を指向する最初の試みの1つではあるが，統合化に成功しているとは言えない．手続き的表現から宣言的表現に至る各種表現形式を真の意味で統合化することが今後の課題である．

## 参考文献


1) Davis, R. and King, J.: An Overview of Production Systems, In Machine Intelligence, 8, eds. by Elcock, E. W. and Michie, D. Wiley (1976).
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) Buchanan, B. G. and Feigenbaum, E. A.: DENDRAL and Meta-DENDRAL: Their Applications Dimension, Artificial Intelligence, 11, pp. 5-24 (1978).
4) McDermott, J.: R1: A Rule Based Configurer of Computer Systems, Artificial Intelligence, 19, pp. 39-88 (1982).
5) Fagan, L.: Knowledge Engineering for Dynamic Clinical Setting: Giving Advice in the Intensive Care Unit, Doctoral Dissertation, Dept. of Computer Science, Stanford University (1979).
6) Griesmer, J. H. et al.: YES/MVS: A Continuous Real Time Expert System, AAAI-84, pp. 130-136 (1984).
7) Shortliffe, E. A.: Computer-Based Medical Consultations MYCIN, American Elsevier (1976).
8) Sacerdoti, E. D.: Planning in a Hierarchy of Abstraction Spaces, Artificial Intelligence, 5, pp. 115-135 (1974).
9) Forgy, C. L.: OPS5 User's Manual, Department of Computer Science, Carnegie Mellon University (1981).
10) 石田 亨：プロダクションシステムと並列処理，情報処理，Vol. 26, No. 3, pp. 213-225 (1985).
11) 小林重信：知識工学の基礎と応用：エキスパートシステム(1)，計測と制御，24-8 (1985).
12) van Melle, W.: A Domain Independent System that Aids in Constructing Consultation Programs, Doctoral Dissertation, Dept. of Computer Science, Stanford University (1980).
13) Rebon, R.: Knowledge Engineering Techniques and Tools in the PROSPECTOR Environment, Rep. No. 243, Al Center, SRI International (1981).
14) Fain, J. et al.: The ROSIE Language Reference Manual, Tech. Note N-1647-ARPA, Rand Corp. (1981).
15) Forgy, C. L.: OPS83 User's Manual, Department of Computer Science, Carnegie Mellon University (1985).
16) Davis, R.: Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases, in R. Davis and D. B. Lenat (eds.), Knowledge Based Systems in Artificial Intelligence, McGraw-Hill (1980).
17) Smith, R. G. and Davis, R.: Frameworks for Cooperation in Distributed Problem Solving, IEEE Trans. on Systems, Man, and Cybernetics, SMC-11, pp. 61-70 (1981).
18) Erman, L. D. and Lessor, V. R.: A Multi-Level Organization for Problem Solving using Many, Diverse, Cooperating Sources of Knowledge, 4-th IJCAI, pp. 483-490 (1975).
19) Lesser, U. R. and Erman, L. D.: A Retrospective View of the HEARSAY-II Architecture, Proc. 5-th IJCAI (1977).
20) Corkill, D. D., Lesser, V. R. and Hudlicka, E.: Unifying Data-directed and Goal-directed Control, AAAI-82, pp. 143-147 (1982).
21) Engelmore, R. E. and Terry, A.: Structure and Function of the CRYSALIS System, 6-th IJCAI, pp. 250-256 (1979).
22) Carhart, R. E. and Smith, D. H.: Applications of Artificial Intelligence for Chemical Inference XX, Computers and Chemistry, 1, 79 (1976).
23) Engelmore, R. S. and Nii, H. P.: A Knowledge-based System for the Interpretation of Protein X-ray Crystallographic Data, Heuristic Programming Project Rep., Hpp-77-2, Dept. of Computer Science, Stanford University (1977).
24) Nii, H. P. and Aiello, N.: AGE (Attempt to Generalize): A Knowledge-based Program for Building Knowledge-based Programs, 6-th IJCAI, pp. 645-655 (1979).
25) Zisman, M. D.: Use of Production Systems for Modelling Asynchronous, Concurrent Process, in Pattern-Directed Inference System (eds. Waterman & Hayes-Roth), Academic Press (1978).
26) 畝見達夫，小林重信：Co-PSs：プロセス概念に基づく分散型プロダクションシステム，情報処理学会第28回全国大会講演論文集 (1984).
27) 小野典彦，小林重信：分散協調型プロダクションシステム Po-PS，計測自動制御学会第3回知識工学シンポジウム講演会資料 (1985).
28) Shubin, H. and Ulrich, J. W.: IDT: An Intelligent Diagnostic Tool, AAAI-82, pp. 290-

295 (1982).
29) Vesonder, G.T. et al.: ACE: An Expert System for Telephone Cable Maintenance, 8-th IJCAI, pp. 116-121 (1983).
30) Nelson, W.R.: REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents, AAAI-82, pp. 296-301 (1982).
31) 都島 功他：ルール型制御方式の実プロセス制御への適用, SICE 学術講演会予稿集, pp. 273-274 (1984).
32) Kowalski, T. and Thomas, D.: The VLSI Design Automation Assistant: Prototype System, 20-th Design Automation Conference (1983).
33) Friedland, P.E.: Knowledge-based Experiment Design Molcular Genetics, Rep. No. 79-771, Dept. of Computer Science, Stanford University (1979).
34) Stefik, M.J.: Planning with Constraints, Rep. No. 80-784, Dept. of Computer Science, Stanford University (1980).
35) Bobrow, D.G. and Stefik, M.: The LOOPS Manual, Xerox PARC (1981).

(昭和 60 年 8 月 28 日受付)