連想・選別型推論のアナロジーによるプロダクションシステムの高速実行方式†

鶴田節夫†† 能見誠†† 宮本捷二††

局所的な知識の追加を繰返すことにより，ソフトウェアの作成が容易に，また段階的にできるプロダクションシステム（略して PS）[1),2)]は，知識工学の実用化への有望なアプローチとして注目されている．しかし，列車運転整理をはじめとするコマンドアンドコントロールシステムなどへ適用しようとすると，複雑・大規模になるうえ，時間的な制約が厳しいため，すべてのプロダクション規則と事実を逐次照合する実行方式，すなわち，従来の原理的な PS（たとえば Newell の PSG）[1),6),11)]の実行方式では，性能上，実用には問題がある．高速化方式も，McDermott らをはじめいくつか提案があるが[5)~8)]，性能・機能上，問題が残されている．これらの問題を根本的に解決するため，前記の原理的な方式に立返って，列車運転整理用の複雑・大規模な PS を実行し，その性能実測結果を分析して，知識の事前整理とその知的な適用による，人間，とくに，熟練者の連想・選別型の高速推論のアナロジーを基本思想とする，PS の（前向き）推論の高速実行方式を提案し，LISP を用いてインプリメントした．VAX 11-780 上で従来方式と比較した結果，PSG のような原理的な方式では 6 時間，McDermott らの高速実行方式[5)]でも 2 分かかった上記の PS が，45 秒で実行できることが実測され，複雑・大規模かつ時間的制約が強い場合にも適用可能な PS の性能を汎用ミニコンで達成できる見通しが高まった．

1. はじめに

知識情報処理における代表的な知識表現・利用方式を提供するプロダクションシステム[1),2)]（略して PS）では，知識の全体や相互関係が不明確でも，必要となる局所的知識の宣言を次々に繰返しさえすればソフトウェアが構築できる．ソフトウェアの拡張・変更は，基本的には，拡張・変更する機能に相当する局所的知識の追加・修正だけを宣言すればよく，知識の全体構造の変更や，従来のプログラムで必要となる制御構造の変更は不要である．そのため，大規模化・複雑化するとともに，例外機能などの追加や変更が多いうえ，高知能化が強く要求されつつある現在および今後の計算機システムの開発，とくに，知識工学の実用化には，PS はきわめて有効である．

一方，PS の実用化には，次のように性能上の問題がある．PS は，プロダクションルール（または，規則）の集合であるプロダクションメモリ（PM）と，事実や仮設（まとめて，事実）の集合であるワーキングメモリ（WM）から成る．規則は，条件部と行動部（結論部）から成る．PS（ただし，前向き推論）では，条件部に設定された条件（条件項）がすべて成立（事実と照合）する規則の集合（競合集合）の中から，最優先の規則が選択されて，つまり競合処理されて実行（適用）され，その行動部の要素（行動項）として指定された処理が順次行われる．どの規則も実行可能でなくなったり，実行しても新事実が生成されなくなったり，行動部に実行終了を指定した規則が実行されれば，推論を終了する．以上のような原理的な PS（たとえば，Newell の PSG）[1),6),11)]の実行方式では，規則の適用時，WM 内の全事実に対して PM 内の全規則の条件項と比較（照合，マッチング）しようとする．そのため，行動部の処理時間が特別に大きい PS を除けば，規則数と事実数が多くなれば，推論時間は本質的に規則数と事実数の積に比例する[6)]☆．また，規則内の変数の値の組合わせの数にも著しく影響される．これは，われわれが適用対象とする，列車運転整理をはじめとするコマンド・アンド・コントロールシステムなど，時間的な制約が強く，規模も大きく複雑なシステムでは，実用上問題である．

このような PS の性能は，適用に際して問題になることが多く，いくつかの高速実行方式が提案されている[5)~8)]．その代表的なものに，音声理解用 PS の高速化

† A Method for Increasing Production System Implementation Efficiency Based on the Associative and Selective Inference Analogy by SETSUO TSURUTA, MAKOTO NOOMI and SHOOJI MIYAMOTO (Systems Development Laboratory, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所

☆ 文献 6) の p. 164, ll. 1~13 によると，PS の 1 サイクルの処理時間は，$T.\ CYCLE \fallingdotseq (P*M*T.\ MCH)+(PA*T.\ ACT)$．ここで，$P$: 規則数，$M$: 事実数，$T.\ MCH$: 1 条件項と 1 事実の照合時間．$PA$: 行動部の項数の平均値，$T.\ ACT$: 1 行動項の実行時間であるが，$PA$ が 5 以下で，$T.\ ACT$ が $T.\ MCH$ と大差のない，つまり，行動部の処理時間（上式の第 2 項）が特別に大きくはない PS では，P と M が数 10 以上，したがって $P*M$ が 1,000 以上の実用規模になれば，条件部の処理時間（上式の第 1 項）に対し第 2 項は無視できるため，推論時間は本質的に $P*M$ に比例する（$T.\ MCH$ は一定と考えてよい）．

を目的として Hayes-Roth, F. が提案した ACORN[5] や，McDermott らの高速実行方式[6] がある．ところが，ACORN は，たとえば，行動項が一つ，条件項が二つなど，直接処理できる規則の型に種々の制約がある．したがって，もとの規則を変換する必要が生じ，このため規則数が増加するうえ，余計な(ダミー)規則さえ追加される．これは，実行効率の低下を招くばかりでなく，PS の長所である，規則を単位とした，モジュール性，リーダビリィティ，説明機能 (Self-explanatory)[1]，を損なう．そのうえ，ACORN は，専用のマルチプロセッサ上での実行方式であり，汎用ミニコンなどにそのまま実装できないので，コストパーフォーマンス・移行性に劣り実用的でないなどの問題がある．一方，McDermott らの高速化方式は，いわゆる弁別ネット[6),8)] を用いて事実に対応する規則を効率よく探索することと，成立した条件数を規則ごとにカウントすることを特徴とする．しかるに，各条件項の構成要素の種類が多くなると弁別ネットの負荷が高くなるうえ，推論実行による事実の追加・削除ごとに，同じ事実を繰返し弁別ネットに通すなど，明らかに実行効率が悪い．さらに，成立した条件数をカウントする方法は，同じ条件が変数の異なる値に対して成立した場合もカウントされるため，条件チェックの効率が悪いなどの問題がある．

以上の問題の適用対象に即した，根本的な解決をはかるため，前記の原理的な方式にまで立返り，適用対象である列車運転整理用の複雑・大規模な PS を実行して，その性能実測データを詳細に分析した．つぎに，その分析結果から，人間，とくに熟練者の「知識を整理し，これを連想的・選択的に利用する効率的な推論」のアナロジーを，PS の高速実行のための考え方のベースとし，これを具体化して，特殊な計算機アーキテクチャや機械依存言語から独立し，PS の長所を損わない，概念的にも整理・体系化された PS 高速実行方式を提案した．

つぎに，本提案方式を，これに最も近いと思われる前記の二つの代表的な高速実行方式と比較し，その相違点，および長所を明らかにした．

最後に，提案方式，および McDermott らの他の高速化方式を，汎用ミニコン VAX11-780 上で，Lisp を用いてインプリメントし，前記の列車運転整理用の大規模，すなわち推論負荷の高い PS を実行した．その結果，原理的な PS (たとえば Newell の PSG[1,),6),11)]) の方式では6時間，McDermott らの高速実行方式[6] では2分，本提案方式では45秒で必要な推論を完了し，本提案方式の有効性を確認した．

2. 高速化の基本思想

2.1 従来方式の性能

表1は，列車の運転整理システムの一部を PS を用いて記述し，前述の原理的な推論方式で実行したときの推論時間，その他を実測したものである．

本システムは終端駅（たとえば新幹線の東京駅）への列車到着遅れの回復案を提案するものである．回復案作成のためのノウハウ的な規則のほか，終端駅の列車発着番線使用のための規則，列車の進行，車両運用に関する規則，ダイヤに関して最低守るべき規則などを，プロダクションルールとしてもつ．推論の前提として入力される事実（初期事実）は，列車の現在位置，到着番線などの設備データ，ダイヤなどである．

発車可能時刻の計算に必要な知識を表現した具体的な規則の例を**図1**に示した．また，**図2**は，本 PS のプロダクションルール（規則）および事実のシンタックスである．本 PS は，ACORN[5] などに比べて，知識の表現がより自然語的なうえ，規則の条件部や行動部の項数などに関する制約もない．

本システムは，規則を先頭から順に探索し，事実と照合した最初の規則を実行し，もし，事実の追加や削除があれば先頭の規則まで戻り，そうでなければ次の規則から，再び探索を続ける．回復案作成用の規則が実行されると，回復案の要素が，新事実として WM

表1 従来の原理的な方式の性能の例
Table 1 Performance data of conventional pure PS implementation method.

総推論時間	21,237秒 (5.9時間)
実行規則探索時間 (条件部の処理時間)	21,199秒 (総推論時間の99.82%)
条件チェック用照合回数	584,641回
条件成立規則に対する条件チェック用照合回数	4,844回 (条件チェック用照合回数の0.83%)
有効探索時間	155秒 (総推論時間の0.73%)
無効探索時間	20,925秒 (総推論時間の98.46%)
規則実行時間 (行動部の処理時間)	37秒 (総推論時間の0.17%)
既知事実チェック時間*	22秒 (総推論時間の0.11%)
ガーベッジコレクション時間	2,896秒 (総推論時間の13.6%)

* 生成された事実が既知事実かどうかをチェックするための時間

発車時刻に関するノウハウ

```
(規則  発車可時刻計算5
 (もし     (番線 (bansen) に列車 (trid) 着)
           (列車 (trid) の 列車種別 は 回送))
 (ならば
  (追加   (あと60秒で番線 (bansen) の列車は発車可)
          (番線 (bansen) に列車 (trid) 停車))
  (削除   (番線 (bansen) に列車 (trid) 着))))
```

PS の規則 ⇩

```
(rule hakei5
  (if
    (>bansen ni >trid chaku)
    (>trid no resshu wa kaiso))
  (then
    (ad (ato 60 byo de >bansen
             no tr wa hasshaka)
        (>bansen ni >trid teisha))
    (rm (>bansen ni >trid chaku))))
```

図 1 列車運転整理用 PS の規則の例
Fig. 1 Example of PS rules for train regulation system.

```
〈規則〉::=(rule スペース 〈規則名〉〈条件部〉〈行動部〉)
〈規則名〉::=〈リスプのアトム〉
〈条件部〉::=(if 〈条件項列〉)
〈行動部〉::=(then 〈追加部〉〈削除部〉)
〈追加部〉::=〈空〉|(ad 〈追加項列〉)
〈削除部〉::=〈空〉|(rm 〈削除項列〉)
〈条件項列〉::=〈条件項〉〈条件項列〉|〈条件項〉
〈追加項列〉::=〈追加項〉〈追加項列〉|〈追加項〉
〈削除項列〉::=〈削除項〉〈削除項列〉|〈削除項〉
〈条件項〉::=〈規則の項〉
〈追加項〉::=〈規則の項〉
〈削除項〉::=〈規則の項〉
〈規則の項〉::=(〈項〉)|(〈項〉)〈規則の項〉
〈項〉::=〈項〉 スペース 〈項〉|〈定数項〉|〈変数項〉|〈関数項〉
〈定数項〉::=〈リスプのアトム〉
〈変数項〉::=〈変数接頭辞〉〈リスプのアトム〉
〈関数項〉::=〈リスプ関数〉
〈変数接頭辞〉::= >
〈空〉::=
〈事実〉::=(〈定数項列〉)
〈定数項列〉::=〈定数項〉 スペース 〈定数項列〉|〈定数項〉
```

図 2 対象 PS の規則および事実のシンタックス
Fig. 2 Syntax of rules and facts in PS studied.
注 1) BNF 記法を使用．リスト間のスペースは許されるが記述は省略した．
注 2) リスプ関数は，事実データに副作用をせず，アトムを値とするものに限る．

に追加されるとともに，回復案リストにも記憶される．回復案生成は，30秒ごとの未来の列車運行状況を推論しながら行う．すなわち，推論開始時点を0秒として30秒ずつ更新される時刻事実をもち，規則を全部探索しても新事実が得られなくなると，時刻事実を更新し，次の時点での回復案の追加と列車運行状況の予測のため，事実に照合する規則を先頭から順次探索し実行する．時刻事実の値などに関する決められた条件に達すれば推論を停止し，回復案リストの内容を表示する．

性能測定は，この時刻条件を6分に（つまり13回，時刻事実が更新されれば推論を停止）して行った．規則数・初期事実数はどちらも約70である．使用した計算機は VAX 11-780，オペレーティングシステムは UNIX（バークレー版），使用した言語は Frantz Lisp である．時間は，Frantz Lisp の Ptime 関数を挿入して測定した．

さて，表1によると従来の原理的な PS 実行方式では列車運転整理案の提案に6時間（CPU タイム）もかかる．実用的な応答待ち時間は長くて1分までだから数100倍以上の高速化が要求される．

2.2 高速化の基本思想

従来方式の PS を，前節の要求に合うように高速化するための基本的な考え方を表1をもとにして導く．

PS の実行時間は，実行規則探索時間と規則実行時間に大別できるが[6]（前章の脚注を参照），表1によると，後者は 0.17%，しかもこれから既知事実チェック（のための照合）時間を除くと 0.06% にすぎない．すなわち，PS の実行時間は，ガーベッジコレクションを別にすると 99.9% 以上が照合時間であることが表1よりわかる．しかも，その大部分が，無効探索時間，つまり，条件と事実を照合した結果，条件が満足されず実行規則が見つからなかった場合の時間である．これから，照合時間（とくに無効探索時間）を0に近づければ，それだけで PS の実行時間は 0.1% になり，最大千倍程度高速化できることがわかる．

さて，人間が規則と事実の2種類の知識（すなわち長期記憶と短期記憶）をもつ[1),11)]のを認めても，記憶している全規則を全事実に対して照合を行って適用可能な規則を選択しているとはとても考えられない．実際，人間，とくに専門家は，与えられた情況に対して関連する規則やノウハウを瞬時に連想し，そのなかから情況に適合するものを効率よく選別，適用し，以後，関連する規則を次々に連想・選別して推論を進めていく．これは関連する事実と規則，規則と規則，換言すれば，同じパターンをもつ知識は瞬時に連想できるようにグループ化，構造化して記憶しているからだと考えることができる．

選別に関しては，連想したノウハウや規則を大局的にチェックしたり，過去の適用結果を覚えていて，事実にマッチしないことが明らかな規則，あるいは，適用しても新事実が得られない規則なども簡単なふるいにかけていると考えられる．

これらの考え方を心理学的に厳密に実証できるかは別にしても，ともかく，このアナロジーを PS の実行方式として計算機上に実現できれば，探索は，全規則，全事実に対してでなく，連想した関連規則，関連事実に対してのみ行えばよい．また，選別により，無効探索時間の削減，規則内の変数の値の組合わせ数に依存する照合時間の低減などが可能となる．こうして，表1によれば 99.9% 以上を占めていた照合時間を，大幅に削減し，最大 1,000 倍程度の高速化が期待できる．

ところで，従来のプログラムでは，ユーザつまりプログラマが，知識の使用順序まで制御フローとして最初から完全に規定する[4]ため，照合による非効率な知識の探索は不要である．しかし，複雑・大規模なソフトウェアでは知識とその使用順序が不明確，煩雑な場合が多いため開発が困難となる．また，制御と知識が分離できないため柔軟性に欠け，保守性・拡張性が低い．フレームやセマンティックネット[2),3)]は，知識と制御は分離しているが，知識の構造を階層やネットワークとしてユーザが指定する必要がある．したがって知識の探索は，指定された構造をたどって効率よくできるが，複雑で大規模なソフトウェアでは知識の全体構造や相互関係が不明確な場合がほとんどであり問題である．PS は局所的な知識（規則）を羅列的に追加していくことにより，ソフトウェアが段階的に構築できるので，複雑・大規模なソフトウェアの開発やプロトタイピング（試験版の早期作成）が容易なばかりか，拡張性・保安性にも優れる．このような長所を損わないで，PS の実行効率を向上させるために，羅列された知識を推論実行前に自動的に構造化しておくことによって，ユーザには制御構造や知識の構造を意識させずに，熟練者の高効率な連想・選別のアナロジーを計算機上に実現することが，本高速化の基本思想である．次章に，その具体的な実現方法，アルゴリズムを述べる．

3. 高速実行方式

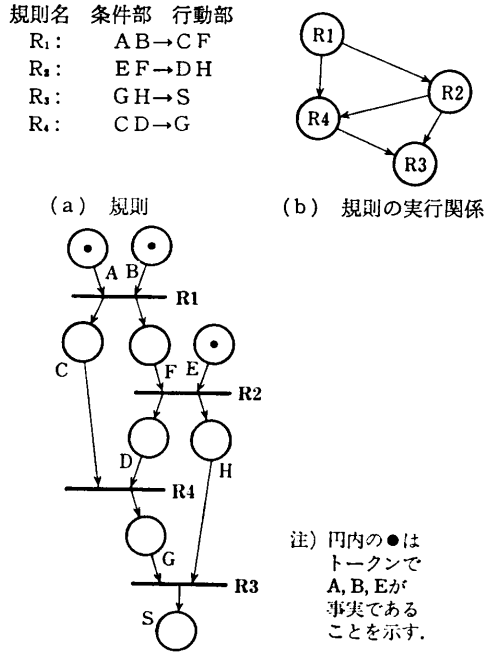
以上の基本思想を具体化しアルゴリズム化することにより，PS の高速実行方式を提案する．まず，関連する知識を瞬時に連想する専門家の推論方式のアナロジーの実現のための知識の事前整理の方法を，つぎに，連想した実行候補規則を効率よく絞るための選別のアナロジーの計算機上での実現方法（実行時フィルタ）について述べる．最後に，これらの方法を一連のアルゴリズムにまとめて PS の高速実行方式として提案する（図6）．

3.1 知識の事前整理

知識の事前整理は，規則の（条件部と行動部の）項を介した規則の連想ネット（図3，図4）を自動構成し，これに事実を埋込んで知識を構造化することにより実現する．

3.1.1 規則の構造化

ユーザが意識して宣言しなくても，プロダクション規則の実行関係は各規則の条件項と行動項によって事前に予測できる．たとえば，**図3**の4規則から成る簡単な PS では，規則 R_1 の実行により，事実C，Fが生成されるから，規則 R_2 と R_4 の実行可能性が事前に予測できる（ただし，R_2 と R_4 が競合集合に属す保証はない）．この実行可能性の関係を連想関係（あるいは連想経路）と呼ぶと，同様に，R_2 から R_3，R_4 へと，R_4 から R_3 への連想関係も存在するから，その全体は，図3（b）のようなネットワークとなる．こ



（a） 規則　（b） 規則の実行関係

（c） 連想ネット

図3 定数項しか含まない規則の構造化

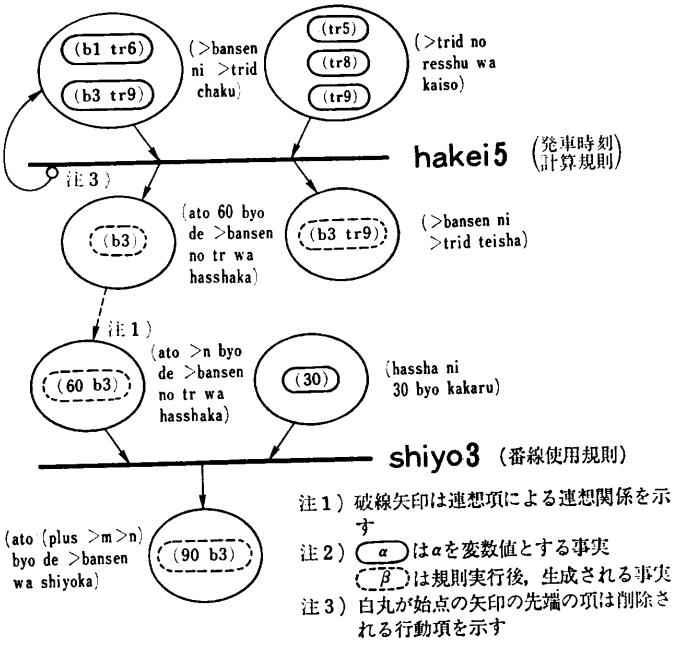
Fig. 3 Structuralization of knowledge without variable terms.

```
(rule shiyo3
 (if
  (ato >n byo de >bansen no
          tr wa hasshaka)
  (hassha ni >m byo kakaru))
 (then
  (ad (ato (plus >m >n) byo de
        >bansen wa shiyoka))))
(rule hakei5  ; ═══ 図1で説明済み ═══
```

(a) PM 中に羅列された規則

```
(b1 ni tr6 chaku)
(b3 ni tr9 chaku)
(tr5 no resshu wa kaiso)
(tr8 no resshu wa kaiso)
(tr9 no resshu wa kaiso)
(hassha ni 30 byo kakaru)
```

(b) WM 中の事実



(c) 連想ネットとして構造化された知識

図 4 知識の構造化（変数を含む場合）

Fig. 4 Structuralization of knowledge with variable terms.

れを，規則の項のレベルまで含めてペトリネット[12]的にモデル化したものが連想ネットである（図3(c)，ただし，**図4**(c)の注3）のように削除を明示しない限り，発火しても入力トークンは削除されない）．A，B，Eが初期事実として与えられると，連想ネットのA，B，Eにトークンが置かれ，まず R_1，以後 R_2，R_4，R_3 の順に発火して，Sにトークンが伝播し，事実Sが生成される．つまり連想ネットをたどるだけで推論ができる．

さて，本提案の対象とするPSは，規則の項が変数や関数を含むため，上例のようにはいかない．そこで規則の項のうち対応する定数項が同じもの（連想項と呼ぶ）を介して連想経路を表現できるように連想ネットを拡張する（図4）．

無関係に羅列された規則を，推論実行時にこのような連想ネットに自動変換して構造化しておけば，McDermott らの方式のように，推論実行時に何度も弁別ネットを使用する[6),8)]ことなく，連想経路をたどってつぎに実行の可能性のある規則（実行候補規則と呼ぶ．競合集合に属すとは限らない．）を効率よく探索できる．変数部のチェックは必要であるが，実行候補規則に対してだけ行えばよく，原理的な方式のように，全規則と全事実に対する定数部をも含めた照合は不要となる．

3.1.2 事実の構造化

前項でも簡単に述べたが，事実はトークンとして連想ネット上に表現できる（図3）．変数や関数を含む場合はこのトークンは，対応する変数の値のリストに拡張される．たとえば，図4(b)の事実 (b1 ni tr6 chaku) と (b3 ni tr9 chaku) は，図4(c)の連想ネットの左上の (>bansen ni >trid chaku) なる項を表す円において，拡張（された）トークン，(b1 tr6) と (b3 tr9) によって表現される．ここで，b1，b3 は変数＞bansen の値，tr6，tr9 は列車名を示す変数＞trid の値を表す．このように，トークン（のリスト）を，規則の項の事実属性として分散記憶することにより，事実が連想ネットに組込まれ，事実の構造化が計算機上で実現できる．

規則の実行（つまり，一種のペトリネット[12]である連想ネットの発火）による拡張トークンの伝播は，対応する変数を介して行う．たとえば図4(c)の項 (ato 60 byo…) のトークン (b3) は前述のトークン (b3 tr9) が変数＞bansen を介して伝播したものである．ところが，初期事実が，図4(b)のように原理的 PS の形式で与えられた場合，これを表現するトークンを連想ネット内に埋込むには，変数に適当な値を与えると初期事実と照合する規則の項を探索する必要がある．これには弁別ネット[6),8)] (discrimination net) を利用する．つまり，規則の項を，**図5**のようなスケルトンを介した連想項のグループに，事前に自動分割しておき，推論開始時，初期事実に対し，そのスケルトン（共通部）を弁別して，連想項を取出し，その中からスケルトン以外の部分（個別部）の照合がとれる規則の項にトークンを埋込む．

規則の項

(>trid no resshu wa kaiso) ……規則の項 Ⓐ
(>trid no resshu wa futsu) ……規則の項 Ⓑ
(>jitr no resshu wa >trkind)…規則の項 Ⓒ

スケルトン(共通部)

(— no resshu wa —) ……スケルトン (共通部)

個別部

(>trid kaiso) …… 規則の項Ⓐの個別部
(>trid futsu) …… 規則の項Ⓑの個別部
(>jitr >trkind) …… 規則の項Ⓒの個別部

連想項

規則の項ⒶとⒷはⒸの連想項，ⒸはⒶとⒷの連想項であるが，規則の項ⒶとⒷは個別部の定数項が kaiso と futsu で一致しないから，互いに連想項とはなりえない．

事実との関係

(tr5 no resshu wa kaiso)…… 上記のスケルトンをもつ事実であり，この事実が存在すれば，Ⓐの事実属性は (tr5) を，Ⓒの事実属性は (tr5 kaiso) を含むが，Ⓑは個別部の定数項が futsu で事実の kaiso と照合しないから，この事実によってⒷの事実属性に (tr5) が加わることはない．

図 5 スケルトン (規則の項の共通部)
Fig. 5 Skeleton: Pattern of terms common among rules.

3.2 実行時フィルタ

選別のアナロジーを実現するための実行時フィルタは連想した実行候補規則の集合（候補集合）から競合集合を得るのに無駄な照合をできるだけ減らそうとするものである．フィードフォワードフィルタとフィードバックフィルタから成る．

3.2.1 フィードフォワードフィルタ

連想経路より得られる候補集合は，定数部に限れば条件項の一つが事実と照合することが保証される規則の集合であるが，これから競合集合を得るには，規則の全条件項に関して，変数部まで含めた照合チェックが必要である．ところが，変数部の照合チェックは，図1の規則 hakei5 の>trid のような同一規則内の複数条件項にまたがる変数の値の一致のチェック (変数値の矛盾チェック) を含むが，この負荷は各条件項の事実属性の要素（トークン）数の積に比例する．たとえば図4の規則 hakei5 は，第1条件項は二つ，第2条件項は三つのトークンをもつ．第1条件項の最初のトークン (b1 tr6) では変数>trid の値が tr6 であるが，>trid の値が tr6 のトークンは第2条件項の事実属性にはない．次のトークン (b3 tr9) では>trid の値が tr9 だから，第2条件項の3番目のトークンがこれに対応する．すなわち，第1条件項の三つのトークンに対しおのおの3回，計6回の変数値の矛盾のチェックが行われる．いま，仮に，規則 hakei5 に第3条件項(>trid no jomuin taiki) が存在したとすると，そのトークンのうち>trid の値が tr9であるものが探索される．もし，第3条件項にトークンが存在しない (事実属性が空) なら，先にこのチェックをしておけば，第1，第2条件項のトークンに対して行った6回の変数値の矛盾のチェックが不要になる．このように，人間の大局的なチェックによる選別のアナロジーとして，事実属性が空の条件項をもつ規則をあらかじめチェックし候補集合から削除して変数の矛盾チェックの負荷を抑えるのがフィードフォワードフィルタである．

フィードフォワードフィルタの具体的な実現方式は次のとおりである．

1) 新事実の追加により候補集合に加えられた各規則に対し，その各条件項の事実属性が空かどうかを調べ，これが空の条件項を一つでももつ規則を候補集合から削除する．

2) 事実の削除により，条件項の事実属性が空になった規則を候補集合から削除する．

3.2.2 フィードバックフィルタ

前項で述べたように，実行候補規則の各条件項の事実属性の要素（トークン）間の変数値の矛盾チェックの負荷は大きい．フィードフォワードフィルタは，実行候補規則の全条件項が定数部に限れば照合する事実をもつことしか保証しないので，候補集合から競合集合を得るには変数値の矛盾チェックが必要である．そこで，過去の経験や判断を記憶しておいて適切な知識を選別する人間や熟練者のアナロジーを用いて，変数部の照合や実行の結果を記憶しておき，同じチェックや実行の繰返しを抑えるのがフィードバックフィルタである．本フィルタは候補集合を絞るためのものであり，競合集合から実行規則を選択する処理(競合処理)ではない．

フィードバックフィルタの具体的な実現方式は次のとおりである．

1) フィードフォワードフィルタにより選別された候補集合に属し，かつ，無効フラグが設定されていないすべての規則に対して変数部の照合チェックを行い，成功すれば，競合集合に加え，不成功なら候補集合から削除する．

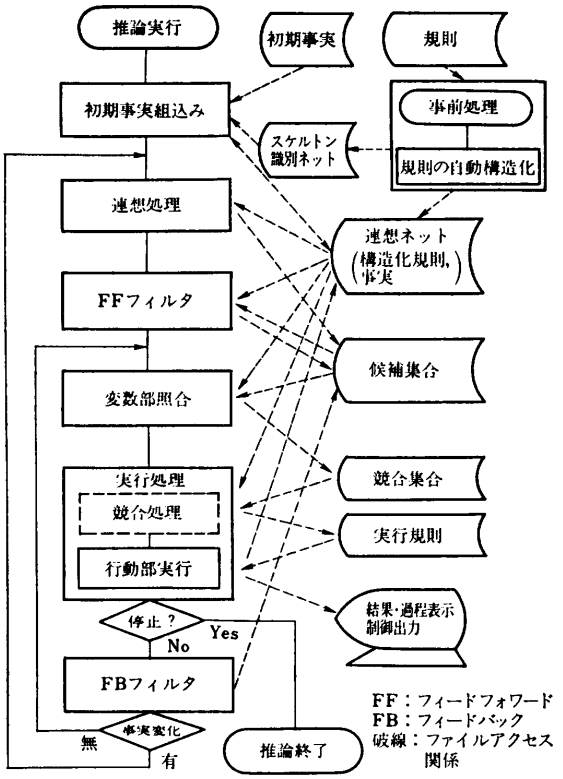


図 6 本提案の PS 高速実行方式
Fig. 6 Control and data flow of proposed efficient implementation method for PS.

2) 規則実行の結果，新事実の追加や既知事実の削除などの変化が PS に生じなければ，この規則の無効フラグを設定する．条件項の事実属性が追加されたり，追加項の事実属性が削除，あるいは削除項の事実属性が追加されたとき，この規則の無効フラグを解除する．

3.3 PS の高速実行アルゴリズム

以上の連想・選別のアナロジーの計算機上での実現方法を PS の高速実行アルゴリズムとしてまとめると次のようになる（**図 6** の流れ図に従って説明する）．

1) 推論実行前処理

a) 全規則に対し，規則の項の共通部（スケルトン）を取出し，規則の項をスケルトンとそれ以外（個別部）の対として記憶する．また，同じスケルトンをもつ項を連想項（属性）として記憶する．さらに，この規則の項を条件部に含む規則を連想規則（属性），追加（削除）部に含む規則を追加（削除）部連想規則として記憶し，全体として連想ネットを構成する（図 3〜5）．

b) 初期事実のスケルトンを識別するための弁別ネット[6),8)]（スケルトン識別ネット）を作成する．

2) 推論実行処理

a) 初期事実組込み

図 4 (b) の形の初期事実をスケルトン識別ネットに通してスケルトンを識別し，このスケルトンを共通部にもち，かつ，個別部の定数部分が照合する規則の項の事実属性の要素（トークン）として，この初期事実を連想ネットに埋込む（図 4 (c) の (＿＿) 内のトークン，すなわち (b1 tr6), (b3 tr9), (tr5), (tr8), (tr9), (30)）．

b) 連想処理

事実属性としてトークンが追加された規則の項の連想規則を候補集合に加える（図 4 (c) の例で規則 hakei5 が実行されると，その行動項 (ato 60…) の連想項 (ato $>n$…) にトークン (60 b3) が追加され，この項を条件項とする規則 shiyo3 が候補集合に加わる）．

c) フィードフォワードフィルタ

新しく追加された実行候補規則の条件項の事実属性をチェックし，一つでも空ならその規則を候補集合から削除する．また，規則実行後，実行規則の削除項およびその連想項のうち事実属性が空になった項の連想規則を候補集合から削除する．

d) 変数部照合，実行処理

無効フラグがオフのすべての実行候補規則について，条件項の事実属性の変数値をチェックし矛盾がなければ，競合集合に加える．競合処理では競合集合から実行規則を取出す．実行時には，変数を介してトークンが伝播する形で，実行規則の追加項とその連想項の事実属性に新事実が追加される（たとえば，図 4 (c) で hakei5 が実行されると，追加項 (ato 60…) にトークン (b3) が，その連想項 (ato $>n$…) にトークン (60 b3) が事実属性として追加される）．事実の削除に関しても同様に，実行規則の削除項とその連想項の事実属性の該当する要素が削除される．

実行停止の規則を実行したり，無効フラグがオフの実行候補規則がなくなれば，推論を停止する．

e) フィードバックフィルタ

変数部照合が不成功の場合，規則を候補集合から削除する．実行によって新事実の追加や既知事実の削除を生じなかった場合，実行規則の無効フラグをオンにする．両場合とも次の実行規則を探すために d) の変数部照合処理に戻る（後者の場合，競合集合が空でなければ変数部照合は行わない）．その他の場合は，候

補集合の追加・削除を行うためb)の連想処理に戻る．また，その事実属性に事実が追加された規則の項の連想規則，削除部連想規則，削除された場合は追加部連想規則の無効フラグをオフ（初期状態）に戻す．

4. 他の高速化方式との相違点

プロダクションシステムの原理的な実行方式の性能を問題として，高速化方式がいくつか提案されている[5)~8)]．ここでは，本方式に最も近いと思われる次の二つの高速化方式と本方式との主要な相違点を述べる．

4.1 ACORN[5)]

F. Hayes-Roth らは，音声理解のための一種のプロダクションシステムの効率的な実行機構 ACORN を提案している[5)]．ACORN は，規則の条件項と行動項をノードとして結合したネットワークを事前にコンパイルしたり，各ノードに事実リスト（instance list)をもたせる点では本提案（厳密には，後述の高速化方式1）に似ているが，おもに以下の点で異なる．

1) ACORN のネットワークで直接表現できる PS の規則は，行動項が一つ，条件項が二つなど型の制約が強く，一般の PS の規則は複数の上記の型の規則に分解して処理するので，等価変換のための余計な規則も心要となり実効効率が低下する．提案方式の連想ネットは，項だけの結合でなく，その上位概念である規則を，項を介して結合したネットワークである．したがって，PS の規則の分解が不要であるから実行効率もよく，推論理由の説明のための適用規則の表示も容易，すなわち，PS の長所を損わない．

2) ACORN には，連想項の考え方やその処理はない．つまり，本提案のほうが，より一般的，自然語的規則をもつ PS の高速化方式といえる．

3) ACORN は，マルチプロセッサでの処理を前提としている．各ノードにプロセッサが割当てられ，条件項に対応するすべてのノードの事実リストが，ACORN に埋込まれたテスト条件を満足するような行動項（に対応する）ノードのプロセッサが，すべて同時に実行される．この意味で ACORN には競合制御はないし，1)で述べた規則の分解などにより，競合制御の機構は複雑で必要以上に効率の悪いものになると考える．また，多数のプロセッサが使用できることもあって，本方式のような実行時フィルタもないし，付加しても，規則の分解などのため効率は悪いと考える．

4.2 弁別ネット

J. McDermott や A. Newell[6)] らは，いわゆる弁別ネット (discrimination net)[6),8)]を用いた PS の高速化方式を提案している．彼等の方式は，事実を入力するとこれに照合する可能性のある条件項をもつ規則を出力する，いわゆる弁別ネットを，事前に作成しておき，推論の実行は，WM に追加（削除）される事実を弁別ネットに入力して，出力された規則のカウンタ（初期値は該規則の条件数）を1減ら(増)し，カウンタが0以下の規則に対してだけ完全な実行条件のチェックを行うものである．彼らの方式と本提案方式を比較すると以下のとおりである．

1) 弁別ネットは，事実と照合する可能性のある項を条件部に含む規則を弁別するのに，規則の項の要素を一つずつ，定数項も含めて照合する．提案方式では，定数部分の照合関係は事前に連想ネットに組込まれているので，推論時は，変数部分だけ照合すればよく，定数項のように変化しない部分の無駄な照合の繰返しは不要である．これにより1回の照合時間が半減することを，5.1 節の2)に実測結果として示す．

また，弁別ネットは，多数の要素と照合をとりその結果により状態遷移を行うことを繰返して弁別を進めるから，処理自体もたんなる照合より複雑で，効率が提案方式の半分以下になることが，やはり実測結果として5.2節の2)の後半部に示される．

2) McDermott らは，カウンタを用いて実行候補規則を絞っているが，カウンタが規則対応にしかないため，規則の特定の項に照合する事実ばかりが生成された場合，他の条件項の事実属性が空でもカウンタが0になり，フィルタとして働かない．人間や熟練者は，カウンタよりむしろ，事実属性が空の条件項をまずチェックすると考えられる．本提案のフィードフォワードフィルタは，このアナロジーである．フィードフォワードフィルタがカウンタ型フィルタの約5倍もフィルタリング効率が高いことを，5.2 節の2）に実測結果として示す．

5. 提案方式の性能評価

本提案の PS 高速化方式について，原理的な方式や他の高速化方式との性能の実験比較・評価を行う．

5.1 原理的な方式との比較評価

表2は，提案方式を VAX 11-780 上に，LISP を用いてインプリメントし，その性能を表1と同様の条件で実測した結果である．これを用いて，原理的な方

表 2 提案方式の性能実測結果
Table 2 Performance data of proposed method.

方式	総推論時間（秒）	条件不成立規則**に対する条件チェック時間（秒）（条件チェック回数（回））	行動部処理時間（秒）
従来の原理的方式	21,237	20,925 (579,797)	35
高速化方式1（知識の構造化）	197	91 (4,497)	28
提案方式*（実行時フィルタ併用）	44	7 (370)	29
高速化方式1＋カウンタ型フィルタ	71	27 (1,907)	33
McDermott らの高速化方式	117	26 (1,907)	80

* 高速化方式2，** 条件チェックの結果，条件不成立が判明した規則
注）表1と同じ列車運転整理用 PS の推論実行結果

式と比較しながら，提案方式の性能を論じる．

1) 従来の原理的な実行方式に比べて，知識の構造化だけによる高速化（高速化方式1）を行った場合100倍，実行時フィルタを併用した場合（高速化方式2，つまり提案方式)では，約500倍性能が向上した．

2) 条件不成立規則（表2を参照）に対する条件チェック回数は，高速化方式1では，原理的な方式と比べて約1/100に，高速化方式2，つまり提案方式では1/1500に減少した．また，その処理時間（無効探索時間）も，従来の原理的な方式と比較して，高速化方式1だけでは1/200に，提案方式では1/3,000に減少した．ここで，無効探索時間（条件チェック時間）の減少率が条件チェック回数の減少率の2倍になるおもな原因は，3.2.2項や4.2節の1）に述べたように，高速化方式1や本提案方式では，事実の分散記憶により定数要素の照合が不要，つまり変数の値だけをチェックすればよいからである．

3) 本提案の高速化方式では，実行候補規則を選択するために連想ネットをたどったり実行時のフィルタリングを行ったりするのは行動部処理時間に含まれるが，このオーバヘッドはほとんど無視できる．たとえば，提案方式（高速化方式2）は高速化方式1と比較して，その行動部処理時間が1秒（総推論時間の0.5%)増加しているが，これが実行時フィルタのオーバヘッドと考えられる．このオーバヘッドは，条件チェック時間を1/10に，つまり84秒（総推論時間の43%）も短縮している実行時フィルタの効果と比べて，十分無視できる．

5.2 他の高速化方式との比較評価

表2の実測結果を中心に，本提案方式と他の高速化方式の性能の比較評価を行う．

1) ACORN は，すでに述べたように PS の規則に対する制約が強く，簡単には比較できないが，等価規則に分解することによる規則数の増加を考えると，性能は，よくて高速化方式1程度，通常その半分以下であると考える．

2) McDermott らの方式に比較して，本提案方式は3倍近く高速である．これは4.2節で述べたように，前者の方式の弁別ネットとカウンタ型のフィルタの非効率さによると考えられる．高速化方式1とカウンタ型フィルタを併用した場合，提案方式に比べて，条件不成立規則に対する条件チェック回数が約5倍，同チェック時間が約4倍，総推論時間が1.6倍もかかっているのが表2よりわかる．これが提案方式の実行時フィルタの効率と McDermott らの方式のカウンタ型フィルタの効率の差であると考えられる．

さらに，McDermott らの高速化方式は，高速化方式1とカウンタ型フィルタを併用した方式に比較して，行動部処理時間が2.5倍（47秒増加），総推論時間が1.6倍かかっている（表2）．WM に追加（削除）された事実に対応する規則を弁別ネットで識別するのは行動部処理に含まれるので，弁別ネットの非効率さの分だけ行動部処理時間が増加したためと考えられる．

6. おわりに

列車の運転整理をはじめとするコマンドアンドコントロールシステムなど時間的制約の強い，実用（中・大）規模のシステムへの適用を目的として，プロダクションシステム（前向き推論）の高速実行方式を提案した．

提案方式は，人間，とくに熟練者が適切な知識を効率よく連想し選別するために行う事前の知識の整理や不完全でも簡単な条件チェックなどによる推論のアナロジーを基本思想とし，知識の連想関係を構造化して事前に自動作成した連想ネットと，これをベースに，推論時に実行候補を絞る実行時フィルタを特徴とする．

提案方式を VAX 11-780 上にインプリメントし，有効性を実験で確認し，上記目的達成の足掛りを得た．

謝辞 本研究の場，およびその方向づけを与えてくださった，日立製作所取締役 三浦武雄コンピュータ

事業本部長，同機電事業本部交通技術本部 大島弘安主任技師，同システム開発研究所 川崎淳所長，同井原廣一副所長，同 春名公一部長に感謝の意を表する．

参 考 文 献

1) 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).
2) Nilsson N.: *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto (1980).
3) 田中幸吉：知能情報処理とロボット特集／1. 総論—知能情報処理とロボットの研究開発動向—，電子通信学会誌，Vol. 65, No. 4, pp. 334-345 (1982).
4) 辻井潤一：人工知能用言語，情報処理，Vol. 22, No. 6, pp. 535-539 (1981).
5) Hayes-Roth, F. and Mostow, D. J.: An Automatically Compilable Recognition Network for Structured Patterns, Proc. 4th Int. Joint Conf. Artificial Intelligence, Tbilishi, USSR, pp. 246-251 (1975).
6) McDermott, J., Newell, A. and Moore, J.: The Efficiency of Certain Production System Implementations, in Waterman, D. A. and Hayes-Roth F. (eds.): *Pattern Directed Inference Systems*, pp. 155-176, Academic Press, New York (1978).
7) Rieger, C.: Spontaneous Computation and Its Role in AI Modeling, in Waterman, D. A. and Hayes-Roth, F. (eds.): *Pattern-Directed Inference Systems*, pp. 69-97, Academic Press, New York (1978).
8) 安西祐一郎他：LISP で学ぶ認知心理学 2 問題解決，東京大学出版会，東京 (1982).
9) 安井 裕：LISP マシン，情報処理，Vol. 23, No. 8, pp. 757-772 (1982).
10) 松崎 稔他(編)：日経データプロ・EDP，製品レポート，日経マグロウヒル社，東京 (1982. 6, 1983. 8).
11) Newell, A.: Production Systems: Models of Control Structures, in Chase, W.(ed.): *Visual Information Processing*, pp. 463-526, Academic Press, New York (1973).
12) Peterson, J. L.: Petri Nets, *ACM Computing Surveys*, Vol. 9, No. 3, pp. 223-252 (September 1977) (斉藤信男(訳)：*bit*, Vol. 10, No. 16, pp. 100-127 (1978)).

(昭和58年11月10日受付)
(昭和60年 2 月21日採録)