解　説

プロダクションシステムの高速化技術†

石田　亨††　桑原和宏††

1. まえがき

現在，エキスパートシステム構築用ツールとして広く用いられているプロダクションシステムは，①EMYCIN に代表される要求駆動型で後向き推論を行うシステムと，②OPS に代表されるデータ駆動型で前向き推論を行うシステムに大別される[小林 85]．ただし，これらのシステムは共にプロダクションシステムと呼ばれるものの，動作原理が著しく異なっており，高速化技術を同列に論じることはできない．本稿では，PSG[Newell 73] に端を発するデータ駆動型のプロダクションシステムの高速化技術を解説する．

データ駆動型のプロダクションシステムは，状況を認識し行動する人間の動作をモデル化したものといわれている[Davis 77]．認識—行動サイクル (recognize-act cycle) を繰り返すこの動作原理を忠実に実現すると，多大な計算機パワーを消費するシステムとなってしまう．したがって，今日までのプロダクションシステムの改良の道筋は，実用的に意味のある応用を記述するために高速化を追求した歴史であったといっても過言ではない．以下では，プロダクションシステムの高速化技術の軌跡を，①条件照合アルゴリズムとその最適化，②推論制御，③言語機能と処理方式の三つの観点から追う．

条件照合アルゴリズムとその最適化方式は本稿の中心的テーマである．現在，標準的に用いられている RETE アルゴリズムを中心に，それ以前あるいは以降に考案されたアルゴリズムや最適化方式を紹介し，それぞれの長所短所，適用領域を検討する．推論制御は，実行するルールの順序や範囲を制御することによって性能向上を図るものである．本稿では，①競合解決時の推論制御と，②ルールやデータの範囲を局所化する注視点制御について述べる．言語機能と処理方式に関しては，プロダクションシステムが商用に供されるようになってからの進歩が著しい．認識—行動サイクルを削減するための言語機能やコンパイル方式，処理系構築手法などを概説する．

なお，既掲載の解説との重複を避けるため，プロダクションシステム全般に関する解説[辻井 79, 小林 85]，及び高速化の一つの手法である並列処理[石田 85] についてはふれないので，他を参照されたい．また，プロダクションシステムの歴史については文献 [安西 86, McDermott 81, Bachant 84, Neches 87] が興味深い．

2. プロダクションシステム

2.1 基本構成

プロダクションシステムはルールを格納するプロダクションメモリ (PM) とルールの実行に必要なデータを格納するワーキングメモリ (WM)，及び PM，WM の双方を参照し解釈実行するインタプリタから構成されている．WM 中のデータはワーキングメモリエレメント (WME) と呼ばれる．

それぞれのルールは，left-hand-side (LHS) と呼ばれる条件要素 (condition element) の連接 (conjunction) と，right-hand-side (RHS) と呼ばれる動作 (action) の集合から構成されている．条件要素は条件に合致する WME が存在すれば成立する*．RHS には LHS 成立時に実行する WME の追加削除を記述する．なお，WME の修正はもとの WME の削除と新しい WME の追加の組み合わせとして実現される．

```
(P Time 0 x                                  ; rule name
   (Goal ↑Type Simplify ↑Object ⟨x⟩) ; LHS
   (Expression ↑Name ⟨x⟩ ↑Arg1 0 ↑Op *)
   -->
   (MODIFY 2 ↑Op NIL ↑Arg2 NIL)) ; RHS
```

図-1　プロダクションルールの例 [Forgy **]

† Art of Performance Improvement in Production Systems by Toru ISHIDA and Kazuhiro KUWABARA (Communications and Information Processing Laboratories, NTT).

†† NTT 情報通信処理研究所

* 現在使用されているプロダクションシステムでは，LHS に正の条件要素 (positive condition element) と負の条件要素 (negative condition element) を記述することができるものが多い．正の条件要素は条件に合致する WME が存在すれば成立する．逆に負の条件要素は条件に合致する WME が存在しなければ成立する．

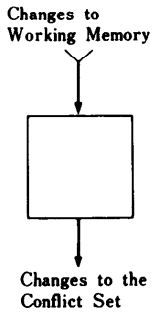

図-2 条件照合処理 [Forgy 82, Miranker 87b]

プロダクションルールの例 (OPS 5[Forgy 81] による記述例) を図-1 に示す.

2.2 動作原理

プロダクションシステムインタプリタは, 以下のサイクル (プロダクションサイクルと呼ばれる) を繰り返し実行する.

① 条件照合 (Match):

すべてのルールについて, LHS とその時点の WM との照合を行いインスタンシエーションを生成する. ここで, インスタンシエーションとは,

<ルール, そのルールの LHS と照合が成功した WME のリスト>

で表されるものである.

② 競合解決 (Conflict Resolution):

インスタンシエーションの中から一つを定めた戦略に従って選び出す.

③ 動作 (Act):

選択されたインスタンシエーションを基にルールの RHS を実行し, WME の追加除去を行う.

より詳細には, 一度実行の対象となったインスタンシエーションを再び実行させないために, 以下の工夫が行われている. まず, インスタンシエーションは競合集合 (Conflict Set) と呼ばれる集合に登録される. 一方動作フェーズでは, 実行したインスタンシエーションをこの競合集合から取り除く. これによって, 同じインスタンシエーションを基に繰り返しルールが実行されることを防いでいる.

言い替えると条件照合フェーズの処理は, 図-2 に示すように前サイクルの WM の変更を基に競合集合がどのように変化するかを計算する処理である. すなわちプロダクションシステムにおいては「条件照合」という用語の意味するところは,「WM の差分に基づく競合集合の差分の計算」である.

3. 条件照合アルゴリズム

3.1 McDermott のフィルタ

PSG[Newell 73] では, プロダクションサイクルごとにすべてのルールについて条件照合処理を行っていたが, これでは同じ処理が繰り返され効率が悪い. そこで, McDermott らはプロダクションシステムの高速化に寄与する知識を分類し, これらを組み合わせて条件照合を効率化するフィルタを構成することを提案している[McDermott 78a].

まず, 高速化に寄与する 3 種の知識を以下に示す.

① Condition Membership: どの条件要素がどのルールに含まれているかに関する知識.

② Memory Support: どの条件要素がどの WME によって満足されているかに関する知識.

③ Condition Relationship: 同一のルールに含まれる複数の条件要素間にどのような関係があるかに関する知識.

フィルタは上記の知識を用いて条件照合を行うべきルールを絞り込むためのものである. フィルタを用いた条件照合は以下の手順で行われる.

① 実行される可能性があるルールをあらかじめ絞り (Filtering Step),

② その後, プロダクションシステムインタプリタが絞り込まれたルールの条件照合を行う (Interpretation Step).

McDermott らは, 一例として以下に示すフィルタを提案しその評価を行っている.

(1) Condition-Membership Filter

図-3 に例を示す. すでに, ルール P 10, P 12, P 13 が条件要素Dを, P 10, P 11 がEを, P 11, P 12 がFをそれぞれ含むことが解析されているとしよう. Filtering Step では条件要素 D, E, F と照合が成功する WME が存在するかどうかを調べることによって (すなわち 3 回の条件照合で), 実行される可能性があるルール (以下これを候補集合と呼ぶ) を導くことができる. 図-3 の場合には, WME にDが存在するので P 10, P 12, P 13 がひとまず候補となり, Eが

```
Production Rule
    P10: (D  E → ……)
    P11: (E  F → ……)
    P12: (F  D → ……)
    P13: (D  D → ……)
Working Memory
    (D F)
```

図-3 Condition-Membership Filter の例 [McDermott 78a]

存在しないので P10 が候補から外される．

（2） Condition-Membership Memory-Support Filter

Memory-Support の最も単純な形態はカウンタである．まずルールごとにカウンタを設けることを考えよう．最初にカウンタにそれぞれのルールの条件要素数を設定する．ルールを構成する条件要素のいずれかと照合が成功する WME が追加されるとカウンタを1減じ，削除されると1加える．カウンタが0以下であればそのルールを候補集合に入れる．この手法と(1)の Condition-Membership Filter を組み合わせると Condition-Membership Memory-Support Filter ができあがる．

一般に Memory-Support は，前サイクルでの条件照合結果をなんらかの形で記録しておくものであるから，おのおののサイクルでの条件照合が大幅に簡略化される反面，WME が追加削除されるたびに記録された内容を更新することが必要となる．したがって Memory-Support が有効であるか否かは，条件照合に要するコスト (matching cost) と記録の更新に要するコスト (update cost) のトレードオフの問題に帰着する．たとえば，前述の例で条件要素ごとにカウンタを設けるようにすれば，より正確なフィルタとすることができるが，記録の更新に要するコストは増大する．

我が国の研究としては，鶴田ら[鶴田 85]があらかじめルールを静的に解析することによって，McDermott のフィルタを改良する手法を提案している．解析結果はどのルールの動作がどのルールの条件要素に影響を与えるかを表すグラフ（一種のペトリネット）となる．このグラフを用いることによって，候補集合の絞り込みを効率化している．

3.2 RETE アルゴリズム

RETE アルゴリズム[Forgy 82] は，OPS 5[Forgy 81], ART[ART 87], KBMS[服部 85], ES/KERNEL[船橋 86], Rule Runner[木下 86] など，多数の商用ツールで用いられている条件照合アルゴリズムである．McDermott らは，前節で述べた3種の知識をすべて用いると Interpretation Step を必要としないフィルタ（すなわち，競合集合を生成するフィルタ）を作ることができると指摘しているが，RETE はまさにそのようなフィルタである．以下にその概要を示す．

RETE アルゴリズムでは，ルールの条件部は RETE

```
(P Plus0x
        (Goal  ↑Type Simplify  ↑Object <N>)
        (Expression  ↑Name <N>  ↑Arg1 0  ↑Op +  ↑Arg2 <X>)
-->     ...)

(P Time0x
        (Goal  ↑Type Simplify  ↑Object <N>)
        (Expression  ↑Name <N>  ↑Arg1 0  ↑Op *  ↑Arg2 <X>)
-->     ...)
```

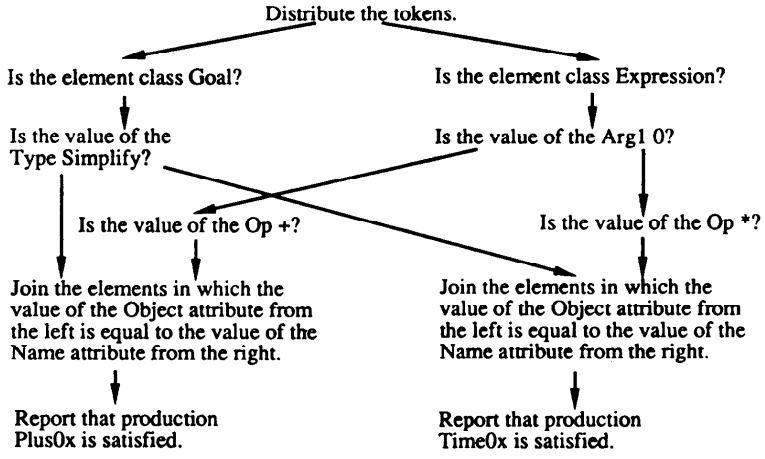

図-4 RETE ネットワークの例[Forgy 82]

ネットワークと呼ばれるデータフローグラフに変換される．動作フェーズで WME が追加されると，その WME（トークンと呼ばれる）が RETE ネットワーク中に流し込まれ，変化にともなうネットワークの更新が行われる．この更新（条件照合フェーズに相当する）は，以下の手順で進められる．まず，LHS の各条件に対して一つの条件内で完了するテスト（intra-condition-test，または selection）が行われ，テストを通過したトークンがネットワーク内（α-memory）に蓄えられる．その後，条件間で変数の値が矛盾しないかどうかを調べるテスト（inter-condition-test，または join）が順に行われ，テストを通過した WME の組がネットワーク内（β-memory）に蓄えられていく．LHS のすべての条件を満たした WME の組（インスタンシエーション）は，RETE ネットワークの終端（terminal）に到達し，競合集合に登録され対応するルールを発火可能とする．RETE ネットワークの更新が完了すると，処理は競合解決フェーズに移行する．

RETE ネットワークの例を図-4 に示す．ネットワークは intra-condition-test を行うネットワークと，inter-condition-test を行うネットワークから構成されている．本解説では前者をパターンネット，後者を join ネットと呼ぶ．パターンネットを構成するノードを 1-input-node, join ネットを構成するノードを 2-input-node と呼ぶ．条件判定をネットワークで表現する手法は，[Hayes-Roth 75, Cohen 78] らによっても研究されているが，RETE の特徴は，むしろ以下の２点にある．

Node Type	R1	XSEL	PTRANS	HAUNT	DAA	SOAR
one-input	3193	1916	1616	874	118	154
α-memory	2366	1432	920	623	91	120
β-memory	3866	1824	738	367	206	295
two-input	5282	2762	1353	1012	285	377
terminal	1931	1443	1016	834	131	103

（1） Number of Nodes

Node Type	R1	XSEL	PTRANS	HAUNT	DAA	SOAR
one-input	122.79	94.04	119.43	80.96	35.39	25.97
α-memory	9.29	6.26	7.20	3.20	2.00	2.58
β-memory	3.03	2.41	3.51	9.28	1.99	8.49
tow-input	36.93	24.03	24.91	35.66	20.31	35.83
terminal	0.96	1.74	1.72	1.51	1.98	3.98

（2） Node Visits per Action

Node Type	R1	XSEL	PTRANS	HAUNT	DAA	SOAR
one-input	8.76	7.00	3.80	5.45	6.69	6.32
α-memory	4.05	3.50	2.84	2.16	3.92	4.60
β-memory	1.39	1.36	1.42	1.29	1.27	1.17
two-input	1.28	1.22	1.26	1.14	1.20	1.15

（3） Structure Sharing

図-5 統計データ[Gupta 83]

① 中間結果の保存（Temporal Redundancy）

サイクルごとに条件判定を繰り返すのではなく，前サイクルでの計算の途中結果をすべてネットワークに保存し，ルールの実行にともなう変更分だけを再計算する．[Gupta 83] による既存の応用システムの調査結果を基に RETE の有効性を検証しよう．図-5（1）は RETE ネットワークのノード数を，図-5（2）は１回の WME の変更によって，再計算されるネットワークのノードの数を表している．このデータから，サイクルごとに再計算されるのはネットワークのごく一部に過ぎないこと，再計算量はプログラムの規模にほぼ独立であること，このためプログラムが大規模になるほどこの手法が有効になることが分かる．

② ネットワークの共有（Structure Sharing）

複数のルール間で，可能なかぎりネットワークのノードを共有させる．図-5（3）にその効果を示す．共有により，α-memory が 1/3〜1/4 に，β-memory が 2〜3 割削減されることが分かる．ネットワークの共有により，実行性能が約 1.4 倍に向上すると報告されている．

3.3 TREAT アルゴリズム

RETE アルゴリズムは中間結果（temporal redundancy）の保存を大きな特徴としていた．しかし，もしルールの実行により前サイクルの中間結果の多くが更新されてしまうとしたら，その保存は意味を失う．それどころか，保存した中間結果を毎回更新するオーバヘッドは耐え難いものとなる．

TREAT[Miranker 87a, 87b] は，以下を骨子とする RETE への対案である．

① β-memory をもたず，サイクルごとに join を計算し直す．（ただし，α-memory はもっている．）

② Join 演算の順序を動的に最適化する．

図-6 にその様子を示す．WME が追加されるとまず追加された WME と他の α-memory（正確には，追加された WME と照合が成功する条件要素以外の α-memory）との間で join が計算される．これを seed-ordering と呼んでいる．join 計算量を削減するためには，一般に数が小さい集合から計算するほうが効率がよいからである．

ルールの発火ごとに WM の内容が大きく変わるような temporal-nonredundant な応用では，TREAT は RETE に比べ明らかに性能がよい．しかし，どの範囲の応用までが TREAT に適するかは明らかではな

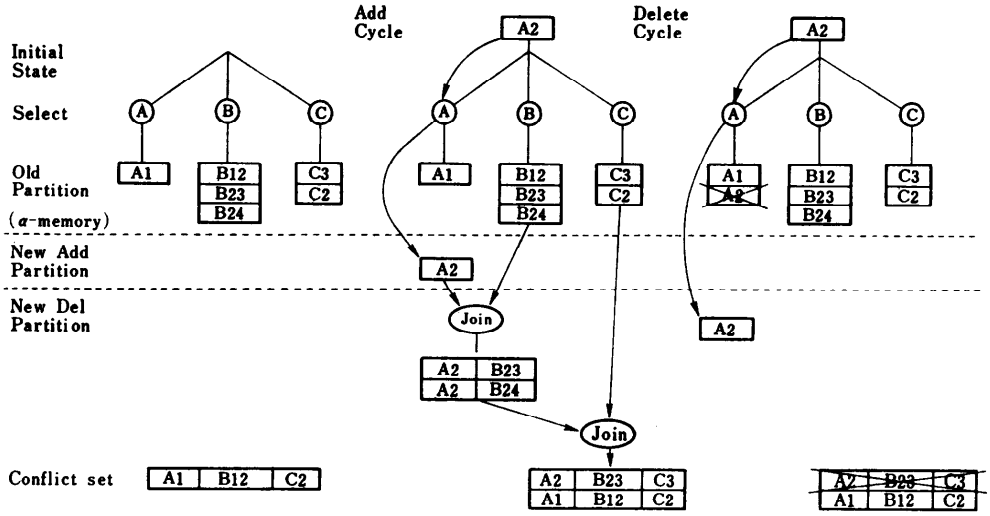

図-6 TREAT アルゴリズム [Miranker 87a]

い．プロダクションシステムの応用が広がるにつれ，他にも RETE アルゴリズムへの批判が出始めている．[Nuutila 87] ではリアルタイム型の応用は temporal-nonredundant で，一般に WME 数は小さいとしている．XC と呼ばれる彼らが C++ 上に開発したプロダクションシステムでは，α-memory さえもたない（すなわち Memory-Support を一切もたない）が，性能は OPS 83 に比べて良いと報告されている．これらの事例は RETE アルゴリズムがあらゆる応用領域に対して万能ではないことを示している．

4. 条件照合の最適化

条件照合アルゴリズムの研究に続いて，条件照合の順序を最適化する研究が行われている．以下では，パターンネットと Join ネットに分けて研究内容を紹介する．

4.1 パターンネットの最適化

パターンネットでは，トークンは通常深さ優先（パターンネットの上から下へ，左から右へ）でテストされる．あるノードでテストが失敗すると未探索の他のルートが試される．葉に相当するノードのテストが成功すると，トークンは対応する条件要素の α-memory に格納され join ネットに引き渡される．

パターンネットを最適化する基本的なアイデアは以下の二つである．

① 可能なかぎり早くテストを失敗させる．

失敗する確率の高いノードを，下から上へ[田野 86]移動させる．この最適化は，プログラムの動作特性を計測した結果に基づいて行われる．

② テスト間の排他性を利用する．

[荒屋 87] では，排他的なテストを排他リンクと呼ばれるポインタで結合する．排他リンクが出ているノードで照合が成功すると，リンク先のノードでの照合は失敗することが明らかであるから，省略することができる．この手法により，1～3割程度性能が向上すると報告されている．

4.2 Join ネットの最適化

OPS5 を含む RETE アルゴリズムのインプリメンテーションの多くは，α，β-memory を単純なリストで実現している．このため，Join 演算が nested loop で計算され，処理時間はデータ量の2乗に比例しているのが現状である．したがって，データ量が増大した場合には Join ネットの高速化が必須となる．これまでに以下の提案がなされている．

（1） ハッシュの利用

[Gupta 87] では，α，β-memory 内のデータをハッシュし Join 演算を高速化している☆．ただし，各 memory 内のデータ量にはばらつきがあるため，memory ごとにハッシュテーブルを用意するのは得策ではない．そこで，ハッシュテーブルを α-memory，β-memory 用の2本にまとめている．ハッシュを用いて OPS5 を試作し評価した結果，join 演算量が比較的多い応用プログラムで約 2.5～3.5 倍程度の性能向

☆ この研究は条件照合の並列処理を目的として行われた．RETE ネットワークのノードごとにプロセスを起動する際に，各プロセスの処理時間を均一化することが本来のねらいである．

```
(p rule1      (p rule2      (p rule3
 when          when          when
 (a)           (a)           (c)
 (b)           (b)           (d)
 ((c)          (f)           (e)
  (d))
 then…         then…         then…
```

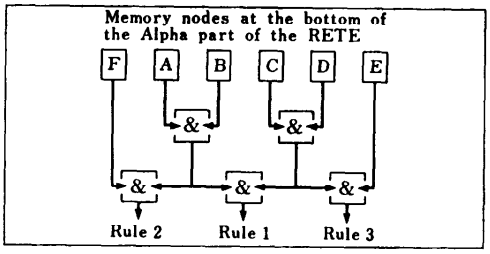

図-7 Join トポロジ[Schor86]

上が得られたと報告されている.

(2) データ間の関係の利用

KBMS[服部 85] では WME のクラス間に関連・グループと呼ばれる関係を明示的に定義することによって, Join 演算の効率化を図っている. たとえば, join を行う WME 間に1対1の関係がある場合にはあらかじめ関連を定義する. これにより WME 間に直接ポインタが張られ, そのポインタを辿ることによって Join の対象範囲を絞り込んでいる.

(3) Join トポロジの指定

ART[ART 87], YES/OPS[Schor 87] などでは, join 演算の実行順序や組み合わせ方法 (Join トポロジ) をユーザが決定する方式を提案している. 図-7 に YES/OPS の例を示す. [Clayton 87] では, 効率のよい join トポロジを決定するために, 次に示すようなヒューリスティクスが提案されている.

① 照合が成功する WME 数が小さな条件要素から join する.

② WME の追加削除によって, 実行時に何度も再計算が必要となる条件要素は後で join する.

③ 類似するルールの join を共通化する. また, これを徹底するため join cluster (複数の条件要素を join し, 改めて join の右入力としたもの) を積極的に利用する (図-7 参照).

①③はプロダクションシステムに限らず, データベースや AI 分野で研究されてきた問い合わせ最適化[Warren 81, Jarke 84, Smith 85] で用いられてきたものと共通である. しかし, ②はプロダクションシステムに特有のヒューリスティクスである. すなわちプロダクションシステムでは, 従来の問い合わせ最適化と異なり, プログラムの動作特性をも考慮しなければならない. ユーザにとっての問題は, これらのヒューリスティクスを個別に用いても性能の改善が保証されない点にある. 性能のよい Join トポロジを見つけることは, 困難な問題であることが指摘されている[Clayton 87].

(4) Join トポロジの最適化

Join 演算の実行時の最適化は, 前述の TREAT のほかに SOAR[Laird 87] でも試みられている. SOAR は chunking と呼ばれる学習機構を用いて実行時に新たなルールを獲得するが, 一般に獲得されたルールは条件用素数が多いため, 最適化を行わないとかえって実行速度に悪影響が生じる場合がある. [Scales 86] では, 獲得したルールを静的に解析して最適化することが試みられている.

実行時の最適化はオーバヘッドが問題となるため, あまり凝った手法を採ることができない. 翻訳時の最適化としては, 筆者らがプログラムの動作特性に基づいて, 複数ルールの Join トポロジを一括して最適化する手法を提案している. 既存の応用システムに適用して, 2倍程度の性能向上が得られている[石田 88].

5. 推論制御

推論制御はプロダクションシステムの実行過程を制御し高速化を図るものである. 以下では, 競合解決と注視点制御を取り上げる.

5.1 競合解決による制御

競合解決戦略の初期の研究は [McDermott 78 b] にみることができる. 競合解決の規則として, 以下のものが提案されている.

① Production Order Rule: 翻訳時に決定されるルールの優先順位である. 条件要素数が多いものを優先するなどの規則がこれにあたる.

② Recency Rule: インスタンシエーションを構成する WME がいつ生成されたか (タイムタグ) に基づいて実行順序を決定する規則である.

③ Distinctiveness Rule: すでに実行されたインスタンシエーションとの類似性 (インスタンシエーションを構成する WME に同じものがあるかなど) に基づいて実行順序を決定する規則である.

④ Special Case Rule: インスタンシエーション間の優先順位を個別に指定する規則である.

ツールとして提供されているプロダクションシステムにはあらかじめいくつかの戦略が組み込まれており, ユーザはその中から適当なものを選択して使用す

る．たとえば，OPS 5[Forgy 81]では，上記の① ②を組み合わせて，LEX，MEA という最新の WME を優先する戦略を提供している．ユーザが独自の競合解決戦略を記述可能なものもある．OPS 83[Forgy 85] や CL[渡辺 86] ではインスタンシエーションの比較を組み込みの関数を用いて定義することができる．

競合解決戦略を巧みに利用してコーディングを行うと，プログラムが簡素化され性能向上に寄与することがある．たとえば，「他のルールが実行されない場合にだけ実行されるルール」を記述するには Production Order Rule を用いると便利である．しかし一方では，プログラム上に現れない暗黙の制御がプログラムの読解性を低下させ，大規模なルールシステムのメンテナンスを困難にすることも指摘されている[Soloway 87]．

競合解決戦略は，プロダクションシステム言語の実行規則であるという側面と，解空間の探索方法を記述するものであるという，二つの側面を有している．OPS 5 を始めとする商用ツールでは，実行規則としての側面が強い．競合解決戦略をルールで記述するものも提案されているが，この場合には探索方法を記述するという側面が強くなる．[Rosenthal 85] では，インスタンシエーションを条件照合対象として，競合解決ルールを記述することによって，次の発火対象となるインスタンシエーションを選択する．問題解決過程から戦略ルールを獲得する研究も数多く行われているが，学習機構の研究に重点がおかれている段階であるため本稿では深く立ち入らない．[Laird 87, Min-ton 87] などを参照されたい．

5.2 注視点制御

ルール数や WME 数が増大した場合には，PM や WM 全体を対象に条件照合を行っていては効率が悪い．

注視点制御は解を得るまでに要する推論の実行サイクル（あるいは実行時間）を最小とするために，対象となるルールやデータを絞りこむ技術である．絞り込みの手法については，たとえば [Hayes-Roth 77] を参照されたい．以下ではむしろ，注視点制御を実現するために役立つメカニズムを紹介する．

（1） PM の注視点制御

OPS 5 プログラムでは，処理フェーズを表す制御用 WME（通常，phase, state, task, context などで表される）によって，実行すべきルールの集合を制御する手法がとられている[Brownston 85]．すなわち，ルールの先頭に制御用の WME をチェックする条件要素を記述することによって，実行対象となるルールとならないルールの切り分けが行われる．OPS 5 以降のシステムでは，ルールセットを導入してルールセット単位での制御移行を可能としているものが多い．ルールセットの性能向上への寄与には以下のものが考えられる．

① 照合範囲をルールセット内に限定できる．

② ルールセットごとに最適な競合解決戦略を指定できる[渡辺 86]．

応答速度への要求が厳しいリアルタイムシステムでは，ルールセットは有効な手法となっている☆．特に [竹中 87, Hsu 87] では①を達成するためにルールセット処理中に生じた WME の変更を，ルールセット終了後にまとめて他のルールセットに反映する方式が採られている．すなわち，あるルールセットによって WME の変更が頻繁に行われても，それが毎回毎回他のルールセットの条件照合対象にならないよう工夫されている．

（2） WM の注視点制御

WME 数の大規模化はプロダクションシステムの性能（特に join の性能）を急速に劣化させる．PRISM[Langley 83] では認知モデルに基づき，WM (short-term memory) とデータベース (long-term memory) を階層化し，活性化されたデータだけを WM に保持するようにしている．PLANET[石田 87]では，意味ネットワークを効率よく探索するために同様の階層化を行い，探索範囲が広がるにつれ自動的に条件照合対象とすべきデータをデータベースから WM に移す方法を採用している．

6. 言語及び処理方式

6.1 言語機能の拡充

言語機能の拡充は，一般には高速化を主眼としたものではないが，プロダクションシステムの場合には，拡充の背景に高速化への強い意志が働いているものが多い．以下では，高速化の観点から機能拡充に関するさまざまな提案を分類する．

（1） 複合ルールの記述

2. で定義したプロダクションシステムは基本的な機能しか備えていない．このため複雑な条件判定をプ

☆ データ駆動型ではないが，PICON [Moore 84] ではルールを 1 次ルール群と，必要が生じた場合にだけ 1 次ルールから起動される 2 次ルール群に分け，きめ細かく注視点制御を行う focus 機能が提供されている．

ログラムしようとすると，複数のルールが WM を介して中間データを授受する処理形態となる．WM の書き換えによるオーバヘッドを削減するためには，複雑な条件を記述できるよう言語機能を拡充しなければならない．POPS 2[広瀬 85]や ART[ART 87] では条件要素間の AND, OR, NOT の任意の組み合わせを記述できるようにしている．EUREKA[田野 87] では，複数の条件要素にまたがる変数間の関係を AND, OR などを用いて記述できるよう拡張が加えられている．また，拡張した言語機能を効率よく処理するための RETE ネットワークの改良方法が提案されている．

（2） 繰り返し処理の記述

基本的なプロダクションシステムでは，繰り返し処理はルールを繰り返し発火させることによって実現される．実行サイクル数を削減するためにはルールを発火させずに LHS で繰り返し処理を行うことを考えなければならない．たとえば，条件記述に全称限量子を導入することが提案されている．ART[ART 87] では，条件要素に FOR-ALL と指定することにより，多数の WME を対象とした条件判定を記述できる．[van Biema 86] でも，セマンティクスはやや異なるが FOR-ALL がプロダクションサイクル数を削減するために導入されている．また別の例としては，繰り返し処理を行うオペレータの導入が考えられている．YES/OPS[Schor 87] では，最大値，最小値を求める maximize, minimize などのオペレータが LHS に記述可能なように導入されている．

（3） 駆動条件の分離

データ駆動型のプロダクションシステムでは，データの変更によってルールが駆動される．ルールの LHS では，この変化をキャッチするために常に条件判定を繰り返しているのであるから，駆動条件の判定 (data-driven match) はきわめてコストが高いということができる．YES/OPS では，駆動条件以外の条件判定 (procedural match) を RHS に移すための言語機能が提案されている．例を図-8 に示す．

```
(p translators2
 when
<g> (goal type: print-translators)
      ; This is the trigger condition
 then
      ; print heading once
  (say Source-language Target-language Person)
  (for-all-matches-of
    (language from: <from-lang> to: <to-lang>)
    (person translate-from: <from-lang>
            translate-to: <to-lang>
                  name: <n>)
  do
    (say <from-lang> <to-lang> <n>)
  )
  (remove <g>)) ;one rule fires, goal removed
```

図-8 駆動条件の分離[Schor 86]

また，駆動条件でよく問題となるのが WME の変更である．基本的なプロダクションシステムで WME の値を変更するためには，元の WME を除去し，変更後の値をもつ WME を新たに追加しなければならない．もし，一つの WME に多くの情報が格納されていると，WME の一部の変更が WME 全体の追加削除を引き起こし，オーバヘッドが大きくなるだけでなく，ルールの冗長な発火が生じることがある．カウンタのようにめまぐるしく値が変化する場合にはなおさらである．YES/OPS では，WME に含まれる値の変更は，WME の追加削除を引き起こさない．RETE ネットワークの更新は，各ノードで WME の変更前と変更後の条件照合結果の差異を調べ，異なる結果が得られたノードに対してだけ行われる．これによってオーバヘッドが軽減されるとともに，言語仕様上からも問題がある，ルールの冗長な発火を防いでいる．

6.2 処理方式の改良

（1） コンパイル方式

OPS 5 では RETE ネットワークのノードをリストなどの中間形式で表現し，インタプリタが解釈実行する方法を用いていた．最近では OPS 83[Forgy 85] などのように，RETE ネットワークを直接手続きに変換する方法が多く採用されている．この場合でも各ノードにおける処理をサブルーチンとして実現しそれを呼び出すか，インライン展開するかで性能が異なってくる．ARCH[中村 87] ではサブルーチン化によって，速度的には 10% 程度遅くなるが，変換後の手続きの規模は 40% 程度小さくなると報告している．また，OPS 5 では実行部も OPS 5 独得のコマンド列で記述され，インタプリタが解釈実行していた．その後の YAPS[Allen 83] をはじめとする多くのシステムでは実行部を直接手続きに変換することによって実行速度の向上を図っている．

RETE ネットワークを直接手続きに変換する方式は性能向上には効果があるが，一方で柔軟性を犠牲にする．たとえば，コンパイル方式にはルールをインクリメンタルにコンパイルするものと，一括してコンパイルするものが考えられる．RETE ネットワークが中間形式で表現されている場合には，ルールの追加削除にともなってインクリメンタルにネットワークを変更することは容易である．しかし，RETE ネットワークが手続きに変換されている場合にはインクリメンタ

ルな変更は困難なものとなる.

(2) 記述言語

OPS5では,当初インタプリタがLispで記述されていたが,その後Blissで再記述され一桁程度性能が向上したと報告されている[McDermott 81]. 現在ではLisp, またはCやPL/1[Cruise 87]などの手続き型言語で記述されている処理系が多い. 以下では,上記以外の新しいパラダイムを用いて記述されたものを紹介する.

Prologを用いて実現された処理系に,POPS2[広瀬85]とKORE/IE[新谷87]がある. Prologを用いると,Prolog処理系が提供する単一化機能やindexingを活用することができる反面,RETEアルゴリズムにおける中間結果の保存を効率よく実現できないという問題が生じる. 実際,KORE/IE[新谷87]ではMemory-Supportを用いずに,どちらかといえばTREATのseed-orderingに近い実現手法が採用されている.

オブジェクト指向パラダイムを用いたものも提案され始めている. 一般に,プロダクションシステムを含むAIツールは発展途上にあり,適用領域が広がるに従って言語機能が拡張される傾向にある. たとえば,条件判定のための言語機能が拡張されると,RETEネットワークのノードとして新たなタイプを導入する必要が生じる. [桑原 88]ではRETEネットワークのノードのタイプをオブジェクトのクラス階層として定義することで,新たなタイプのノードの導入を容易にしている. また,クラス階層を用いてノードの種類を積極的に細分化することによって,処理の専用化を図り性能向上に役立てている. 一方,Opus[Laursen 87]ではSmalltalkを用いてプロダクションシステムが記述されている. 条件照合はSmalltalkのオブジェクトを対象に行われる. LHSやRHSにSmalltalkのメソッドが記述できるなど,プロダクションシステムとオブジェクト指向言語の融合が図られている.

7. むすび

プロダクションシステムの高速化技術を駆け足で紹介した. PSGから今日に至るまでの条件照合アルゴリズムと最適化手法の進歩を明らかにすることに主眼を置いた. また,高速化の観点から推論制御と言語及び処理方式の動向を解説した.

一般に新しいプログラミングパラダイムが定着するには少なくとも10年を要するといわれる. プロダクションシステムも新しいパラダイムであるため,高速化以外にも,テストやメンテナンス手法の確立,データベースや他のプログラミングパラダイムとの融合,ATMSや学習機構の導入など,さまざまな課題を抱えている. 本稿がプロダクションシステムの理解と研究の発展に役立てば幸いである.

謝辞 本研究の機会を与えていただいた村上国男知識処理研究部長,吉田清主幹研究員,中野良平主幹研究員に感謝します. また村山隆彦氏をはじめ討論いただいた知識処理研究部の方々に感謝します.

参考文献

[Allen 83] Allen, E.: YAPS: A Production Rule System Meets Objects, AAAI-83, pp. 5-7 (1983).

[安西 86] 安西祐一郎: OPS5と私, Computer Today, Vol. 5, No. 13, pp. 4-9 (1986).

[荒屋 87] 荒屋真二, 百原武敏, 田町常夫: プロダクションシステムのための高速パターン照合アルゴリズム, 情報処理学会論文誌, Vol. 28, No. 7 (1987).

[ART 87] ART Version 3.0 Reference Manual, Inference Corp. (1987).

[Bachant 84] Bachant, J. and McDermott, J.: Rl Revisited: Four Years in the Trenches, AI Magazine, Fall, pp. 21-32 (1984).

[Brownston 85] Brownston, L., Farrell, R., Kant E. and Martin, N.: Programming Expert System in OPS5: An Introduction to Rule-Based Programming, Addison-Wesley (1985).

[Cohen 78] Cohen, B.L.: A Powerful and Efficient Structural Pattern Recognition System, Artificial Intelligence, 9, pp. 223-255 (1978).

[Clayton 87] Clayton, B.D.: ART Programming Tutorial, Volume Three: Advanced Topics in ART, Version 3.0, Inference Corp. (1987).

[Cruise 87] Cruise, A., Ennis, R., Finkel, A., Hellerstein, J., Klein, D., Loeb, D., Masullo, M., Milliken, K., Van Woerkom, H. and Waite, N.: YES/L1: Integrating Rule-Based, Procedural, and Real-time Programming for Industrial Applications, 4th IEEE Conference on Artificial Intelligence Application, pp. 134-139 (1987).

[Davis 77] Davis, R. and Jonathan, K.: An Overview of Production Systems, Machine Intelligence, Vol. 8, pp. 300-332, J. Wiley and Sons, New York (1977).

[Forgy 81] Forgy, C.L.: OPS5 User's Manual, Technical Report CS-81-135, Department of Computer Science, Carnegie Mellon University (1981).

[Forgy 82] Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, Vol. 19, pp. 17-37 (1982).

[Forgy 85] Forgy, C. L.: OPS 83 User's Manual and Report, Production Systems Technologies, Inc. (1985).

[船橋 86] 船橋誠壽, 増位庄一, 高橋 栄, 磯辺 寛, 林 利弘: ES/KERNEL と EUREKA-Ⅱの推論高速化技術, 日経コンピュータ, 9.15, pp. 121-131 (1986).

[Gupta 83] Gupta, A. and Forgy, C. L.: Measurement on Production Systems, Technical Report CS-81-135, Deparment of Computer Science, Carnegie Mellon University (1983).

[Gupta 87] Gupta, A., Forgy, C. L., Kalp, D., Newell A. and Tambe, M.: Results of Parallel Implementation of OPS 5 on the Encore Multiprocessor, Technical Report CS-87-146, Department of Computer Science, Carnegie Mellon University (1987).

[服部 85] 服部文夫, 清水信昭, 土屋秀幸, 桑原和宏, 和佐野哲男: 知識ベース管理システム (KBMS), 情報処理学会, 知識工学と人工知能研究会, 41-6 (1985).

[Hayes-Roth 75] Hayes-Roth, F. and Mostow, D. J.: An Automatically Compilable Recognition Network for Structured Patterns, 4 th IJCAI, pp. 246-251 (1975).

[Hayes-Roth 77] Hayes-Roth, F. and V. R. Lesser.: Focus of Attention in the HEARSAY-II Speech Understanding System, 5 th IJCAI, pp. 27-35 (1977).

[広瀬 85] 広瀬紳一: 強力なパターンマッチング機能を持ったプロダクションシステム記述言語 POPS 2, 日本ソフトウェア科学会, 知識プログラミングシンポジウム資料, KP-85-8 (1985).

[Hsu 87] Hsu, C., S. Wu and J. Wu: A Distributed Approach for Inferring Production Systems 10 th IJCAI, pp. 62-67 (1987).

[石田 85] 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).

[石田 87] 石田 亨: データ駆動型プロダクションシステムによる意味ネットワークの探索, 情報処理学会論文誌, Vol. 28, No. 10 (1987).

[石田 88] 石田 亨, 横尾 真: プロダクションシステムのトポロジカル・オプティマイザ, 情報処理学会, 知識工学と人工知能研究会, 56-7 (1988).

[Jarke 84] Jarke, M.: Common Subexpression Isolation in Multiple Query Optimization, in Query Processing in Database Systems (Kim, W., Reiner, D. and Batory, D. Eds.), Springer, New York, pp. 191-205 (1984).

[木下 86] 木下哲男, 三樹弘之: ルール型知識システム構築用ツール Rule Runner の開発, 沖電気研究開発, Vol. 53, No. 4, pp. 59-64 (1986).

[小林 85] 小林重信: プロダクションシステム, 情報処理, Vol. 26, No. 12, pp. 1487-1496 (1985).

[桑原 88] 桑原和宏: オブジェクト指向に基づくプロダクションシステムの実現, WOOC '88, 日本ソフトウェア科学会 (1988).

[Laird 87] Laird, J. E., Newell A. and Rosenbloom, P. S.: SOAR: An Architecture for General Intelligence, Artificial Intelligence, 33, pp. 1-64 (1987).

[Langley 83] Langley, P.: Exploring the Space of Cognitive Architectures, Behavior Research Methods and Instrumentation, 15, pp. 289-299 (1983).

[Laursen 87] Laursen, J. and Atkinson, R.: Opus: A Smalltalk Production System, OOPSLA '87, pp. 377-387 (1987).

[McDermott 78 a] McDermott, J., Newell A. and Moore, J.: The Efficiency of Certain Production Implementations, in Pattern Directed Inference Systems (Waterman D. A. and Hayes-Roth, F. Eds.), Academic Press, pp. 155-176 (1978).

[McDermott 78 b] McDermott, J. and Forgy, C. L.: Production System Conflict Resolution Strategies, in Pattern Directed Inference Systems (Waterman, D. A. and Hayes-Roth, F. Eds.), Academic Press (1978).

[McDermott 81] McDermott, J.: Rl The Formative Years, AI Magazine, Summer, pp. 21-29 (1981).

[Minton 87] Minton, S. and Carbonell, J. G.: Strategies for Learning Search Control Rules: An Explanation-based Approach, IJCAI-87, pp. 228-235 (1987).

[Miranker 87 a] Miranker, D. P.: TREAT: A Better Match Algorithm for AI Production Systems, AAAI-87, pp. 42-47 (1987).

[Miranker 87 b] Miranker, D. P.: TREAT: A New and Efficient Match Algorithm for AI Production Systems, TR-87-03, University of Texas (1987).

[Moore 84] Moore, R., Hawkinson, L., Knickerbocker C. and Churchman, L.: A Real-time Expert System for Process Control, 1 st IEEE Conference on Artificial Intelligence Applications, pp. 569-576 (1984).

[中村 87] 中村 明: プロダクションシステム ARCH-2 の処理系概要と性能評価, 第 1 回人工知能学会全国大会, pp. 193-196 (1987).

[Neches 87] Neches, R., Langley, P. and D. Klahr,: Learning, Development, and production Systems, in Production System Models of Learning and Development (D. Klahr, P.

Langley and R. Neches, Eds.), The MIT Press (1987).

[Newell 73] Newell, A.: Production Systems: Models of Control Structures, in Visual Information Processing (Chase, W. G. Ed.), pp. 463-526, Academic Press (1973).

[Nuutila 87] Nuutila, E., Kuusela, J., Tamminen M., Veilahti, J., Arkko, J. and N. Bouteldja: XC—A Language for Embedded Rule Based Systems, SIGPLAN Notices, Vol. 22, No. 9, pp. 23-32 (1987).

[Rosenthal 85] Rosenthal, D.: Adding Meta Rules to OPS 5—A Proposed Extension, SIGPLAN Notices, Vol. 20, No. 10, pp. 79-86 (1985).

[Scales 86] Scales, D. J.: Efficient Matching Algorithms for the SOAR/OPS 5 Production System, STAN-CS-86-1124, Stanford University (1986).

[Schor 86] Schor, M. I., Daly, T. P., Lee, H. S. and Tibbitts, B. R.: Advances in RETE Pattern Matching, AAAI-86, pp. 226-232 (1986).

[新谷 87] 新谷虎松: 推論エンジン KORE/IE, Logic Programming Conference '87, pp. 233-242 (1987).

[Smith 85] Smith, D. E. and Genesereth, M. R.: Ordering Conjunctive Queries, Artificial Intelligence, 26, pp. 171-215 (1985).

[Soloway 87] Soloway, E., Bachant J. and Jensen, K.: Assessing the Maintain-ability of XCON-in-RIME: Coping with the Problem of a VERY Large Rule-Base, AAAI-87, pp. 824-829 (1987).

[竹中 87] 竹中一起, 横井玉雄: プロセス制御向エキスパートシステム構築ツール(2)—高速推論機構—, 情報処理学会第 35 回全国大会, pp. 1723-1724 (1987).

[田野 87] 田野俊一, 増位庄一, 坂口聖治, 船橋誠壽: 知識ベースシステム構築用ツール EUREKA における高速処理方式, 情報処理学会論文誌, Vol. 28, No. 12, pp. 1255-1268 (1987).

[辻井 79] 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).

[鶴田 85] 鶴田節夫, 能美 誠, 宮本捷二: 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, 情報処理学会論文誌, Vol. 26, No. 4, pp. 696-705 (1985).

[vanBiema 86] van Biema, M., Miranker, D. P. and Stolfo, S. J.: The Do-loop Considered Harmful in Production System Programming, 1 st International Conference on Expert Database Systems, pp. 125-138 (1986).

[Warren 81] Warren, D. H. D.: Efficient Processing of Interactive Relational Database Queries Expressed in Logic, 7 th VLDB, pp. 272-281 (1981).

[渡辺 86] 渡辺正信, 岩本雅彦, 山之内徹, 出口幸子, 松田裕幸: CL におけるルール指向プログラミング, 情報処理学会, 知識工学と人工知能研究会, 46-3 (1986).

(昭和 63 年 2 月 10 日受付)