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前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 83]。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工の動作をモデル化したものでいわれている[Davis 77]。認識-行動サイクル (recognize-act cycle) を繰り返すと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いる。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えても過言ではない。以下では、プロダクションシステムの高速化技術の特徴を、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解説する。

条件照合アルゴリズムとその最適化方式は、本稿の中心的なテーマである。現在、標準的で用いられている RETE アルゴリズムを中心に、それ以前あるいは現在に考察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融合推論時の推論制御、②ルールやデータの範囲を局所化する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、既報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 79、小林 83]、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。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) の関連 (conjunction) と、right-hand-side (RHS) と呼ばれる動作要素 (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) (Goal (Type Simplify (Object (x))); LHS (Expression (Name X) (Argl 0 (Op =)) --)) (MODIFY 2 (Op NIL, (Arg2 NIL)); RHS)

図-1 プロダクションルールの一例[Power 88]

† Art of Performance Improvement in Production Systems by
Toru ISHDIA and Kukhiro KUWABARA (Communications
and Information Processing Laboratories, NTT).
†† NTT 情報通信研究機構

††† 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。正の条件要素と正の条件要素と 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。
プロダクションルールの例（OPS5,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、WM の追加除去を行う。

より詳細に示すと、一度実行するタスクは、インスタンシエーションを再び実行せずに、以下の次のタスクが行われるもの。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ている。

言い替え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 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の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込む（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に示す。すなわち、ルール P10, P12, P13 が条件要素 D を、P10, P11 が E を、P11, P12 が F とそれぞれ含むことが解析されているよう。Filtering Step では条件要素 D, E, F と照合が成功する WM 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（以下これを候補ルール呼ぶ））を導くことができる。図-3の場合は WM に D が存在するので P10, P12, P13 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 P 10 が候補から外される。

(2) Condition-Membership Memory-Support Filter

Memory-Support の最も単純な形式はカウンタである。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 減し、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。この方法と(1)の Condition-Membership Filter を組み合わせると Condition-Membership Memory-Support Filter ができる。

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、条件照合要するコスト（matching cost）と記録の更新要するコスト（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られるが、記録の更新要するコストは増大する。

我が国の研究としては、鶴田ら(1988)が考えた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現するグラフ（一種のベクトリネット）となる。このグラフを用いることによって、候補集合の枚記込を効率化している。

3.2 RETE アルゴリズム

RETE アルゴリズム(Foray 87) は、OPS5(Foray 81), ART(Art 87), KBM5(原田 85), ES/KERNEL(船越 86), Rule Runner(木下 85) など、多数の実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インタプリテーションステップ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>)

(...)

Distribute the tokens.

Is the element class Goal?

Is the element class Expression?

Is the value of the Type Simplify?

Is the value of the Arg1 0?

Is the value of the Op +?

Is the value of the Op *?

Join the elements in which the value of the Object attribute from the left is equal to the value of the Name attribute from the right.

Report that production Plus0x is satisfied.

Join the elements in which the value of the Object attribute from the left is equal to the value of the Name attribute from the right.

Report that production Time0x is satisfied.

図-4 RETE ネットワークの例(Foray 81)
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され、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（条件無条件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) に蓄えられていいく。LHS のすべての条件を満たした WME の組 (インスタンシェーション) 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] によく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。

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数の数を表している。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とこの手法が有効になることがわかる。

② ネットワークの共有 (Structure Sharing)

複数のルール間で、可能な限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 割削減されることを示す。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、その保存は意味を失う。それでも、保存した中間結果を毎回更新するオーバヘッドは避けたいものとなる。

TREAT [Miranker 84, 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 との他の 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ない。
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。
[Retina 87] ではリアルタイム型の応用は temporal-nonredundant で、一般に WME 数は小さいとい
じされている。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 さえもな
い（すなわち Memory-Support を一切もない）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、バ
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、トークンは通常選ばれる（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ノードが試される。新たに適用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句の 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 可能なうえで大きく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 [Wright 86] 移動させると、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。
[Short 87] では、互換的テストを排他リンクと呼ばれるボインタで結合する。排他リンクが示している
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

OPS 5 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
実装している。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と便宜は
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 OPS 5 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ものである。
（2）データ間の関係の利用

KBMS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。ただし、Joinを行うWME間で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間で直接ポインタが張られ、そのポインタを巡ることによってJoinの対象範囲を絞り込んでいる。

（3）Joinトポロジの指定

ART、YES/OPS、Schorなどでは、Join演算の実行順序や組み合わせ方（Joinトポロジー）をユーザが決定する方式を採用している。図7にYES/OPSの例を示す。[Clayton 87]では、効率のよいJoinトポロジーを決定するために、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Joincluster（複数の条件要素をJoinし、改めてJoinの右入力としたものを）を積極的に利用する（図7参照）。

図7はプロダクションシステムに限らず、データベースやAI分野で研究されている問い合わせ最適化で用いられているものと共通である。しかし、図7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トポロジーを見つけることは、困難な問題であることが指摘されている[Clayton 87]。

4. Joinトポロジーの最適化

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と呼ばれる学習機能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変数数が多いため、最適化を行わない場合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ーヘッドが問題となるため、あまり凝った手法を採用することができない。翻訳時の最適化としては、筆者がプログラムの動作特性に基づいて、複数ルールのJoinトポロジーを一括して最適化する手法を提案している。既存の実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[石井 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。条件変数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。
ボラジョンシステムの高速化技術

Vol. 29 No. 5

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して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 照会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 [浦辺 85]。

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 87 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法が採られている。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 の大規模化はボラジ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 87] 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[Gray 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、探索範囲がかか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、一般的には高速化を主眼としたものではないが、ボラジ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. で定義したボラジョンシステムは基本的な機能しか備えていない。このため複数の条件判定をプ

データベース構造ではないが、PICCIN (Moore 87) ではルールを 1 次ルール群ごと、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、それが順次注視点制御を行う force 機能が提供されている。
プログラムしようすると、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ーヘッドを削減するために、複雑な条件を記述できるような言語機能を拡充しなければならない。POPS2[56]や ART[87]では条件要件間の 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[57]では、複数の条項要件と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
基本的なプログラマーショングシステムでは、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、条件記述に全称量化子を導入することが提案されている。ART[87]では、条件条項に 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断を記述できる。[van Biema 86]でも、サンプリングははやや異なるが FOR-ALL がプログラマーショング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[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 に記述可能のように導入されている。

（3）動詞条件の分離
データ動的型のプログラマーショングシステムでは、データの変更によってルールが動動される。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行っているのであらかじめ、動詞条件の判定 (data-driven match) は行われてコストが高いということができる。YES/OPS では、動詞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。

また、動詞条件で逐次問題となるのが WME の変更である。基本的なプログラマーショングシステムで 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、一回の 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、オーバーヘッドが大きくなるだけでなく、ルールの元長な発火が生じることがある。カウンタのように求められる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や、WME の追加削除を取り起こさない。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、異なる結果を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ーヘッドが軽減されるとともに、言語構文からも問題がある。ルールの元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
OPS5 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 OPS5[Forbus 83]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。このような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きれを呼び出す、インタライイン実行するかで性能が異なってくる。ARY[86]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% 程度小さくなくなると報告している。また、OPS5 では実行部と OPS5 独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 YAP[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タプ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ある。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タプ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タプリメンタル
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
ていたが、その後Blissで再記述され--桁程度性能が
向上したと報告されている[McDemott 82]。現在では
Lisp、またはFやPL/I[Cruse 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、記入以外
の新しいパラメータ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[阿部
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
活用することが可能である[RETIE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を用いないに、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メータを用いたことも提案さ
れ始めてきている。一般に、プロダクションシステムを含
むAIツールは発展途上であり、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、DTS
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 82]ではRETIE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
を微細に分化することによって、処理の単一化を図り
性能向上に役立っている。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。条件構成は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. まとめ

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
を置いていた。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メータが定着する
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メータであるため、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メータとの融合、
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、吉田清平研究員、中野美平研究員
に感謝します。また村山隆彦氏をはじめ諸氏
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

Rule System Meets Objects, AAAI-83, pp.5-7
(1983).

[安西 86] 安西祐一郎：OPS5と私、Computer

[荒屋 87] 荒屋真二、百原武敏、町田常夫： プロダ
クションシステムのための高速パラメータ照合アル
ゴリズム、情報処理学会論文集、Vol.28, No.7

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

[Bachant 84] Bachant, J. and McDemott, J.: RI
Revisited: Four Years in the Trenches, Al

[Brownston 86] Brownston, L., Farrell, R., Kant
E. and Martin, N.: Programming Expert Syst-
em in OPS 5: An Introduction to Rule-Based
Programming, Addison-Wesley (1985).

[Cohen 78] Cohen, B.L.: A Powerful and Ef
ficient Structural Pattern Recognition System,

[Clyaton 87] Clayton, B.D.: ART Programming
Tutorial, Volume Three: Advanced Topics

[Cruse 87] Cruse, A., Ennis, R., Finkel, A.,
Hellerstein, J., Klein, D., Loeb, D., Masullo,
M., Miliken, K., Van Woerkom, H. and Waite,
N.: YES/L 1: Integrating Rule-Based, Pro-
cedural, and Real-time Programming for Indu-
strial Applications. 4th IEEE Conference on
Artificial Intelligence Application, pp. 134-139

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

Technical Report CS-81-135, Department of
Computer Science, Carnegie Mellon Univer-


[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.


