知識工学と人工知能　49－4
（1986. 11. 11）

知識の排他性を利用したパターン照合アルゴリズム

荒屋　真二　　百原　武敏　　田町　常夫
福岡工業大学

純粋なプロダクションシステムではパターン照合が推論コストの大部分を占める。この照合コストを削減するために，Reteアルゴリズムはプロダクション間の類似性に関する知識と，各プロダクションが作業記憶の内容によって現在どの程度満足されているかについての知識をうまく活用している。本論文は上記２種類の知識に加えて，プロダクション間の排他性に関する知識と，照合成功確率に関する知識を利用した，より強力なパターン照合アルゴリズムを提案し，Reteアルゴリズムにはかなり無駄な照合が残されていることを明らかにする。また，ＯＰＳ５と同様の文法を持つプロダクションシステム記述言語を16ビットパーソナルコンピュータ上に実現し，三つのサンプルプログラムによって提案アルゴリズムの有効性を実験的に示す。

A FAST PATTERN MATCHING ALGORITHM USING KNOWLEDGE OF EXCLUSIVENESS

Shinji ARAYA , Taketoshi MOMOHARA and Tuneo TAMATI
Fukuoka Institute of Technology
3-30-1, Wajirohigashi, Fukuoka-shi, 811-02, Japan

Pattern matching spends the greater part of inference cost in pure production systems. In order to reduce the matching cost, the Rete algorithm makes good use of knowledge about similarities among productions together with knowledge of the degree to which each production is currently satisfied by the contents of working memory.

This paper proposes an augmented pattern matching algorithm which uses, in addition to the above two kinds of knowledge, knowledge of exclusiveness among productions and knowledge of success probability of each pattern matching. A production system with almost the same syntax as OPS5 has been implemented on 16 bits personal computer and the effectiveness of the proposed algorithm is experimentally shown by three sample programs.

1。まえがき

純粋なプロダクションシステムの推論効率は良くないので、それを改善するための手法がいくつか提案されてきた[1)-7)]。しかし、まだ処理時間の点から実時間制御システムや対話型システムなどへの適用には限界があり、推論の高速化は重要な研究課題となっている[6)7)]。

プロダクションシステムの推論効率化をねらった従来研究は、その着眼点により次の三つに大別できるだろう。

（１）無駄な照合の削減による効率化[1)2)5)]

（２）コンパイラによる効率化[5)]

（３）並列処理による効率化[3)6)7)]

これらのアプローチは相互に密接な関連をもつとともに、併用も可能である。事実、研究が最も進んでいるプロダクションシステム記述言語ＯＰＳでは、上記(1)、(2)のアイデアがインプリメントされており[3)5)]、(3)の研究が現在行われている[6)7)]。

本稿では、知識工学の立場から最も興味深く、かつ本質的な意味での効率化ともいえる上記(1)の問題を考察の対象とする。無駄な照合を減らすために提案されている具体的方法は次の２点に集約できる。

（１）現在の照合結果を保存しておくことにより、同一の照合の繰り返しを避ける。

（２）プロダクション間の類似性に関する知識を利用して、同一の照合をまとめて処理する。

ここで、照合結果をどの程度の詳細度で記憶しておくかによって、さらには、知識の類似性をどの程度の詳細度で考えるかによって種々のアルゴリズムが実現できる[2)]。

現在最も効率的といわれているReteアルゴリズム[5)]も上記の考えに立脚しているが、まだ無駄な照合が行われている。筆者等は、これら二つのアイデアに加え、知識の排他性と知識の利用頻度に関する知識を利用したパターン照合アルゴリズムをもつプロダクションシステムをパソコン上に実現し、推論効率がReteアルゴリズムより向上することを確認したので報告する。

2。対象とするプロダクションシステム

本稿で取り上げたプロダクションシステムは、OPS5[4)]とほぼ同様の文法をもつ。すなわち、各プロダクションはプロダクション名、条件部、動作部からなり、さらに条件部は条件要素から、動作部は動作項から構成される。条件要素はクラス名のあとに属性名と属性値のペアが０個以上続く。属性名は記号“^”が前置されることで識別され、属性値としては定数と変数がある。変数は“<”と“>”で囲まれることで識別される。属性値の前では“>”（より大きい）や“<=”（以下）などの範囲を指定する６種類の述語と、データ型が等しいことを表わす述語“<=>”を使用できる。さらに選言命題や連言命題もある制約のもとで使用できる。動作項では、make，remove，modifyなどを含む計15種類の動作を記述できる。これらの文法に関する詳細かつ厳密な内容は文献４）を参照されたい。

作業記憶は作業記憶要素の集合である。作業記憶要素の形式は条件要素とほぼ同様である。ただし、属性値として変数、述語、選言および連言命題を使用できないこと、クラス名の前にタイムタグと生成プロダクション名が付けられることが相違している。

条件部を構成するすべての条件要素と作業記憶内の作業記憶要素のどれかとの照合が成功するとき、そのプロダクションは適用可能となる。適用可能なプロダクションの集合を競合集合とよぶ。ある競合解消戦略にもとづいて競合集合の中から一つのプロダクションが選択され、その動作部が実際に実行される。その結果として作業記憶の内容が更新される。プロダクションシステムはこのようなプロダクションの選択・実行を繰り返しながら（認知・行動サイクル）推論を進める。本稿で提案するパターン照合アルゴリズムは、それぞれの認知・行動サイクルにおける競合集合を効率的に求める方法である。

3。照合アルゴリズム

3.1 基本的考え方

Reteアルゴリズム[5)]は次の二つの基本的考え方を生かして競合集合を効率的に求めている。

(1) 現在の作業記憶に対する照合結果を条件要素単位以上で保存し、次の認知・行動サイクルでの照合に活用する。これにより、変化しなかった作業記憶要素と条件要素（条件要素はすべて変化しない）との照合を再度やり直すという無駄を排除できる。

(2) 条件要素間の部分的共通性をあらかじめ抽出しておき、同一の照合をまとめて行うようにする。これにより、同一の照合を何度も繰り返すという無駄を回避できる。

上記(1)において、「照合結果を条件要素単位以上で保存し」とあるが、この意味は条件要素を構成するクラス名や属性名と属性値のペアのすべての照合が成功したときにのみ保存されるということであり、一部分が成功してもその結果は保存されない。また、同一プロダクションの二つ以上の条件要素が満足しているときには、個々の条件要素が満足しているという情報のほかに、それら複数個が同時に満足しているという情報も別途独立して保存される。同一プロダクションを構成するすべての条件要素が満足している状態はこの特殊な場合であり、この情報も前述の競合集合という形で別途保存される。

さて、本稿で提案する照合アルゴリズムでは、上記二つに加えて、次のような考え方が導入されている。

(3) ある作業記憶要素に対して、ある条件要素のクラス名の照合が成功したならば、異なるクラス名をもつ他の条件要素との照合は必ず失敗に終わる。また、ある属性値の照合が成功したときには、クラス名が同じでも対応する属性値が異なっている条件要素との照合は必ず失敗する。このような知識の排他性をあらかじめ抽出しておき、実行時に利用すれば明かに失敗する照合を回避することができる。

(4) 前述の排他関係にあるクラス名や属性値が複数個存在する場合、なるべく早い時点で照合が成功するように照合順序を工夫してやれば、より多くの照合失敗を避けられる。

3.2 条件部の構造化と照合処理

条件要素とプロダクションとの対応づけはネットワークとして表現可能である。次の二つのプロダクションを例として考えよう。

```
(P P1 (C1 ^A11 V11 ^A12 V12)
      (C2 ^A21 V21 ^A22 V22)
  -->
      (...) ...)
(P P2 (C2 ^A21 V21 ^A22 V22' ^A23 <X>)
      (C3 ^A31 V31 ^A32 <X>)
  -->
      (...) ...)
```

これら二つのプロダクションに対する最も単純で自然なネットワーク表現は図1のようになるだろう。根ノードは条件要素の数だけ分岐し、その先のノードで分岐することはない。根ノードと端末ノード（図中のP_1とP_2）以外のノードは、一つのリンクが入り込む1入力ノードと二つのノードが入り込む2入力ノードに分けられる。1入力ノードではクラス名や属性値の照合が行われ、2入力ノードでは対応する二つの条件要素が同時に満足しているかがチェックされる。もし、二つの条件要素間で同一の変数が使われているときには、それらが同一の値に束縛されているかがチェックされる。ある作業記憶要素がどの条件要素との照合に成功するかは、その作業記憶要素をトークンとして図1のネットワークに流してやればよい。1入力ノードでの照合に成功したトークンは次のノードへ送り出され、失敗したトークンはバックトラックして別の条件要素に対応するパスを流れ出す。もしあるトークンが2入力ノードに到達したときには、対応する条件要素との照合が成功したことを意味する。

さて、前節(1)の考え方によれば、2入力ノードにノードメモリを設け、そこへ到達したトークンを保存する。左側のリンクから入って来たトークンは左メモリに、右側のリンクからのトークンは右メモリに格納することによって、そのトークンがどの条件要素と照合に成功したかを識別できるようにする。このようにして現在の作業記憶との照合結果のすべてを2入力ノードに保存しておくならば、次の認知・行動サイクルにおいては、作業記憶の変化分、すなわち新たに追加あるいは削除された作業記憶要素だけをトークンとして流せばよいことになる。この場合、追加された作業記憶要素には“+”を付け、削除されたものには“−”を付けたトークンを考えることにより、両者の違いを識別できるようにする。マイナストークンがノードメモリに到達したときには、それと同じ内容をもつプラストークンがノードメモリの中から消去される。

次に前節(2)の考え方を導入すると、図1のネットワークは図2のようになる。これがReteネットワークと呼ばれているものであり[5]、同じ照合を行う1入力ノードがまとめられている。その結果として、

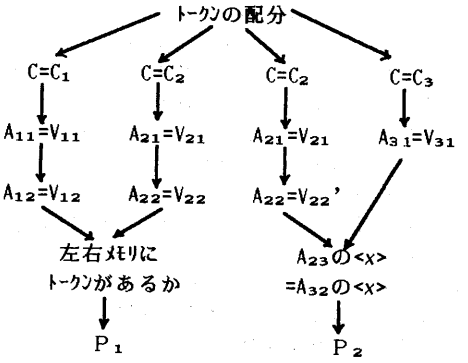

図1　単純ネットワーク

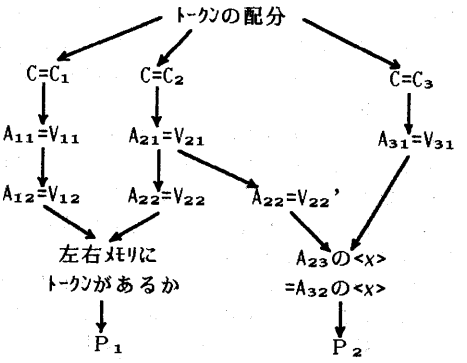

図2　Reteネットワーク

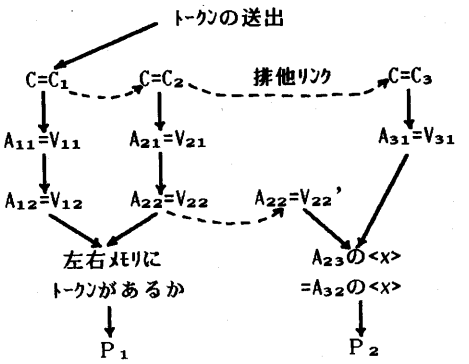

図3　排他ネットワーク

Reteネットワークのノード数は単純ネットワークのそれより減少する。トークンの流れ方やノードメモリの使用方法は単純ネットワークの場合と全く同じである。

さらに、前節(3)の考え方を導入すると、図2のネットワークは図3のようになる。図3では、排他関係にある1入力ノードが排他リンク（図中の破線）で連結され、他のリンクと区別されている。排他リンクが出ているノードでは、そこでの照合が成功したときには排他リンクはないものとみなされ、その先でのバックトラックによってトークンが排他リンクの方へ流れることはない。照合が失敗したときにのみ排他リンクへトークンが送出される。これ以外の点はReteネットワークの場合と同じである。

最後に、前節(4)の考え方も反映したネットワークを考える。排他関係にあるノードは排他リンクにより連結され、上流にあるノード（トークンが先に流れるノード）ほど早い時点で照合が行われる。ゆえに、照合成功の確率が高いノードを上流にもってこれば照合失敗回数を減らすことができる。この照合成功確率は、各ノードでの照合に成功するトークンの発生頻度に比例する。この発生頻度は問題状況によって変化するので、あらかじめ厳密に算出することはできない。

そこで、ここでは次のような簡便な方法をとる。つまり、条件要素の中に出現する回数の多い排他ノード（排他関係にあるノード）ほど照合成功の確率が高いと仮定し、出現回数の多い順に排他ノードを並べ換える。これはあくまでもヒューリスティックな方法であるが、次節においてその効果が確認される。このような排他ノードの並べ換えを行うと図3は図4のようになり、これが本稿で提案するネットワークである。

3.3 動作例

本節では前節の提案ネットワークによる照合処理方法を前述の例を用いて説明するとともに、照合回数が単純ネットワークやReteネットワークの場合よりいかに減少するかを示す。

今、すべてのノードメモリが空の状態で、次の六つのトークンが連続して生成された場合を考える。

T_1 = +(C_1 ^A_{11} V_{11} ^A_{12} V_{12})
T_2 = +(C_2 ^A_{21} V_{21} ^A_{22} V_{22})
T_3 = +(C_2 ^A_{21} V_{21} ^A_{22} V_{22}' ^A_{23} V_x)
T_4 = +(C_3 ^A_{31} V_{31} ^A_{32} V_x')
T_5 = +(C_3 ^A_{31} V_{31} ^A_{32} V_x)
T_6 = -(C_2 ^A_{21} V_{21} ^A_{22} V_{22})

まず、トークンT_1（以下では単にT_1と書く）は図4の根ノードから流れ出し、最初の1入力ノードでクラス名の照合が行われる。クラス名はC_2でないので照合に失敗し、T_1は排他リンクに送出される。再びクラス名の照合が行われるが、今度はそれに成功するので$C=C_3$につながる排他リンクはないものとみなされ、もう一方のリンクへT_1が送出される。次のノードでは属性A_{11}の値がV_{11}であるかの照合に成功し、T_1は次のノードへ引き渡される。そこでは属性A_{12}の値がV_{12}であるかの照合がなされ、それに成功する。次にT_1は2入力ノードに右側のリンクを通って到着し、右のノードメモリに格納される。この2入力ノードでは単に左のノードメモリの中にトークンがないかどうかがチェックされ、何もないので照合処理は完了する。

ひき続いてT_2が根ノードから送出され、クラス名の照合、属性A_{21}の値の照合、属性A_{22}の値の照合のすべてに成功して、2入力ノードに左側のリンクを通って到着する。ここでは、T_2が左のノードメモリに格納されてから、右のノードメモリにトークンが

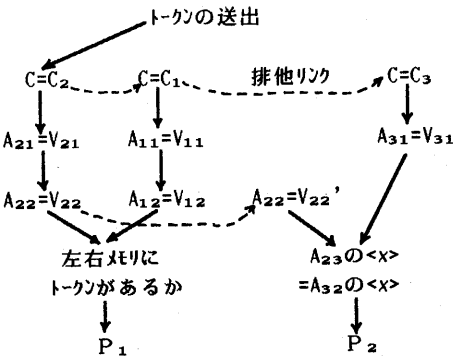

図4　提案ネットワーク

存在するかが調べられる。今度はT_1が存在しているので、この2入力ノードは次の合成トークンを生成

して端末ノードへ送出する。

$$T_{12}=+[(C_2\ \hat{}A_{21}\ V_{21}\ \hat{}A_{22}\ V_{22})\ (C_1\ \hat{}A_{11}\ V_{11}\ \hat{}A_{12}\ V_{12})]$$

ここで注意を要することは、２入力ノードから合成トークンが送出されても、左右のノードメモリには合成前のトークン（T_1およびT_2）を残しておくということである。これによって条件要素単位での照合状態を保持するわけである。合成トークンT_{12}が到着した端末ノードは、T_{12}の符号が＋であるのでプロダクション名P_1とT_{12}を競合集合に追加する。T_2はこれまでに二つの分岐ノードを通過してきたが、そこから出るもう一方のリンクは両者とも排他リンクであった。ゆえに、T_2はバックトラックする必要がないので照合完了となる。

次のトークンT_3、T_4、T_5は条件部に同一変数＜X＞が二つ存在するプロダクションP_2が競合集合に加えられる過程を示すためのものである。まず、T_3は前と同じような照合処理を行われて図４の右側の２入力ノードの左メモリに格納される。同様にしてT_4は同じ２入力ノードの右メモリに格納される。属性A_{23}の変数＜X＞にはV_xが束縛され、属性A_{32}の変数＜X＞にはV_x'が束縛されている。ゆえに、同一変数＜X＞に異なる値が束縛されているので、この２入力ノードは合成トークンを生成せず照合を完了する。次のT_5は同様にして同じ２入力ノードに右側のリンクを通って到達し、その右メモリに格納される。今度は、T_5の属性A_{32}の値と左メモリにあるT_3の属性A_{23}の値が同一であるので照合が成功し、次の合成トークンが生成される。

$$T_{35}=+\ [(C_2\ \hat{}A_{21}\ V_{21}\ \hat{}A_{22}\ V_{22}'\ \hat{}A_{23}\ V_x)\ (C_3\ \hat{}A_{31}\ V_{31}\ \hat{}A_{32}\ V_x)]$$

T_{35}を受けた端末ノードはプロダクション名P_2とT_{35}のペアを競合集合に追加する。

最後のトークンT_6は、マイナストークンに対する処理を説明するためのものである。マイナストークンの場合にも２入力ノードに達するまではプラストークンと同様の処理が行われる。ゆえにT_6はT_2と同じ処理を受けて図４の左側の２入力ノードに左側のリンクを通って到達する。２入力ノードはT_6がマイナストークンなので、同一の内容をもつプラストークンを左メモリの中から探し、T_2を見つけ出す。そして、T_2を消去するとともに、次の合成トークンを生成し送出する。

$$T_{62}=-[(C_2\ \hat{}A_{21}\ V_{21}\ \hat{}A_{22}\ V_{22})\ (C_1\ \hat{}A_{11}\ V_{11}\ \hat{}A_{12}\ V_{12})]$$

このT_{62}は以前にこの２入力ノードから送出されたT_{12}と符号を除いて全く同一である。ゆえに端末ノードは競合集合に以前に加えたP_1とT_{12}のペアを競合集合の中から削除する。以上でT_1～T_6に対する照合処理は終了する。新たに発生するトークンに対しても同様の処理が実行される。

図３のネットワークを用いた場合の照合処理も同じ考え方で実行される。排他リンクを含まない図１および図２のネットワークを用いた場合には、途中で分岐したときの処理方法だけが異なる。すなわち、分岐ノードでは、トークンのコピーが分岐数だけ作られて送出される。並列処理機能のない通常の計算機では、コピーを作らずにとりあえず一つのリンクにトークンを流し、途中で照合が失敗するか、あるいは端末ノードに到達した時に、バックトラックして分岐点に戻り、もう一方のリンクに流してやるようにする。

図１～図４のネットワークを用いたときのそれぞれの照合過程を表１にまとめた。ノードメモリの初期状態は空であり、前述のT_1～T_6のトークンがこの順序で発生した場合を想定している。照合に成功した場合をＳ、失敗した場合をＦで表わしている。ただし、根ノードおよび端末ノードでは照合を行わないので何も書いていない。また、２入力ノードのどちらかのノードメモリにトークンが到達した場合、他方のノードメモリにトークンが存在しない時にはＦとしている。ここでの例ではプロダクション数がわずか二つであるが、知識の構造化を進めるに伴って照合回数が次第に減少することがわかる。もう少し細かく見ると、知識の共通性を利用することは成功と失敗の両方の照合回数を減少させるが、知識の排他性ならびに利用頻度を用いることは照合失敗の回数だけを減らすことに効果があることがわかる。

表１ 各アルゴリズムにおける照合過程

トークン	単純	Rete	排他	提案
T1	SSSFFFF	SSSFFF	SSSF	FSSSF
T2	FSSSSSSFF	FSSSSFF	FSSSS	SSSS
T3	FSSFSSSFF	FSSFSFF	FSSFSF	SSFSF
T4	FFFSSF	FFSSF	FFSSF	FFSSF
T5	FFFSSS	FFSSS	FFSSS	FFSSS
T6	FSSSSSSFF	FSSSS	FSSSS	SSSS
照合回数	46 (21)	35 (16)	30 (11)	28 (9)

（S:照合成功、F：照合失敗、括弧内はFの数）

４．試作システムの概要

前章で述べた照合アルゴリズムを用いたプロダクションシステム記述言語処理系を試作した。そのソ

フトウェア構成を処理の基本的流れがわかるように書くと図５のようになる。本処理系は大きく分けると（１）ネットワークコンパイラ、（２）推論エンジン、（３）コマンドインタープリタの三つから構成される。

コンパイラは外部記憶装置に格納されているプログラム（プロダクションの集合）、あるいはキーボードから直接入力されたプログラムをＳ式に変換し、条件部をコンディションメモリに、動作部をアクションメモリに格納する。次に、コンディションメモリの情報を、照合アルゴリズムに対応したネットワークの形に変換し、ネットワークメモリに格納する。本処理系では、図２～４に対応したネットワークを

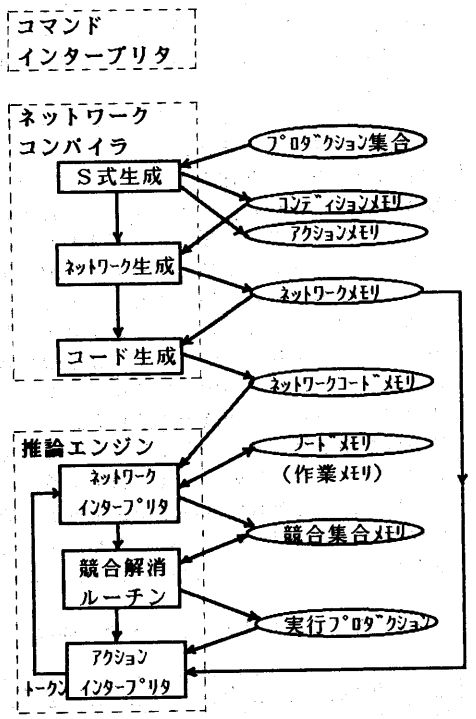

図５ システム構成と処理フロー

生成することができ、ＯＰＳ５にはない新らしいコマンド optimize により構造化レベルを選択できるようにした。最後に、このネットワークは線形化[5]されてネットワークコードとなる。このコードはＯＰＳ５とかなり異なっており、図２～４の三つの構造化レベルに対応できる内部表現になっている[11]。このネットワークコンパイラは前処理として知識ベースの構造化を行う部分であり、一度実行すれば知識ベースに変更が加えられない限り再実行の必要はない。

推論エンジンは実際に推論計算を行う部分で、知識の構造化が済んだあとでのみ実行可能である。まず、ネットワークインタープリタは、生成されたトークンに対してネットワークコードを順次解釈しながらパターン照合処理を行う。そのトークンがある条件要素を満足した時、すなわち２入力ノードに到達した時にはノードメモリの内容を更新する。さらに、そのトークンによって新たなプロダクションが成立したり、これまで成立していたプロダクションが不成立になった時には競合集合メモリの内容も更新される。競合集合は競合解消戦略にもとづいてソートされた形で保存される。ゆえに、競合集合メモリの更新は競合解消ルーチンが行う。実行プロダクションは常に競合集合メモリの先頭に存在している。

コマンドインタープリタはユーザの入力したコマンドを解釈して対応する処理ルーチンに制御を渡す。未サポートのコマンドも若干あるが、逆にＯＰＳ５にないコマンドも準備されている[11]。

なお、本処理系はMS-DOS版Turbo Pascalで記述された。

５．実験的性能評価

5.1 実験内容

Reteアルゴリズムと提案アルゴリズムの性能を比較するためにサンプルプログラムを用いて処理時間を計測した。サンプルプログラムは(1)列車機器故障診断[10]、 (2)ハノイの塔[8]、(3)モンキーとバナナ[9]の三つである。これらを用いた理由は、単にＯＰＳ５の文法で書かれたプログラムのリストが入手できたからにすぎない。それぞれのアルゴリズムおよびプログラムに対して、処理時間をコンパイル時間と実行時間の二つに分けて計測した。ここでいうコンパイル時間とは、プロダクションを構造化してネットワークコードを生成するのに要する時間である。実行時間とは、実際の推論計算に要する時間である。使用計算機はPC-9801Fである。

5.2 結果と考察

実験結果は表２のようになった。提案アルゴリズムを用いた場合のコンパイル時間は、Reteアルゴリズムの場合より大きいが、これはコンパイル時に知識の排他性の検出と出現頻度によるノードの並べ替えが行われているためである。しかし、実行時間は逆に提案アルゴリズムの方が小さくなっており、コンパイルに時間をかけた効果がここに現れている。このように、コンパイル時間と実行時間との間にはトレードオフ関係が存在する。実行時間の減少率がサンプルによってばらついているが、これは知識の排他性を利用することの効果が、知識ベースに内在する排他性の量に依存しているからである。サンプルの中では列車機器故障診断プログラムが知識の排

他性を一番多く含んでいるといえる。

表2 計算時間の比較(機種:PC-9801F)

ﾌﾟﾛｸﾞﾗﾑ名	列車機器故障診断	ﾊﾉｲの塔 ﾃﾞｨｽｸ=3	ﾓﾝｷｰとﾊﾞﾅﾅ
ﾌﾟﾛﾀﾞｸｼｮﾝ数	85	8	27
Reteｱﾙｺﾞﾘｽﾞﾑ	11.42	2.53	7.00
(A)	57.54	7.76	7.38
提案ｱﾙｺﾞﾘｽﾞﾑ	33.42	2.55	7.63
(B)	40.41	6.73	6.84
増減率(%)	292.64	100.79	109.00
(B/A*100)	70.23	86.73	92.68

(上段:ｺﾝﾊﾟｲﾙ時間、下段:実行時間、単位:sec)

本処理系は現在のところTurbo Pascalで記述されているため、プログラムおよびデータのアドレス空間が64KBと小さい。ゆえに、プロダクション数が100を越えると、個々のプロダクションが大きい場合には処理が不可能になるのが現状である。しかし、これは本質的な問題ではなく、より大きなデータ領域を扱えるLattice Cなどで書き換えれば解決される。

6. あとがき

本稿では、知識の排他性ならびに知識の利用頻度を活用したパターン照合アルゴリズムを提案し、Reteアルゴリズムにおける無駄な照合を削減できることを示した。

今回試作した処理系はTurbo Pascalで記述したため取り扱えるプロダクション数をこれ以上増やす余裕はあまりない。ゆえに、未サポートのコマンドの整備も含めて現在C言語で書き換えを進めている。今後の課題には、知識ベースの開発効率に大きな影響を及ぼすコンパイル時間の短縮化がある。プロダクションがわずか一つ追加された時でも始めから知

参考文献

1) Hayes-Roth,F. and Mostow,D.J.: An Automatically Compilable Recognition Network for Structured Patterns, Proc. of 4th Int. Joint Conf. of Artif. Intell., pp.246-251 (1975)

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

3) Fogy,C.L.:On the Efficient Implementation of Production Systems, Ph.D.Thesis, Carnegie-Mellon University (1979)

4) Fogy,C.L.:OPS5 User's Manual, Department of Compt. Sci., Carnegie-Mellon University (1981)

5) Fogy,C.L.:A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem, Artif.Intell., Vol.19, pp.17-37 (1982)

6) Fogy,C.L. et al:Initial Assessment of Architectures for Production Systems, AAAI-84 (1984)

7) 石田: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)

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

9) 吉村:ルールベース・エキスパートシステム構築ツールの基本形OPS5、日経コンピュータ別刷、pp.73-88 (1985)

10) 匹田、西内、荒屋:列車機器故障診断システムのOPS5によるプロトタイピング、第5回シミュレーション・テクノロジー・コンファレンス発表論文集、pp.23-26 (1985)

11) 百原、安富、荒屋:OPS5のパーソナルコンピュータ上での実現、福岡工業大学研究論集、第19号