プロダクションシステムにおけるマッチング候補の概念を用いた高速パターンマッチングアルゴリズム

松下光範[†] 馬野元秀[††,☆]
鳩野逸生[††] 田村坦之[††]

Reteアルゴリズムをはじめとする従来のプロダクション・システムの高速化手法では，ルールの条件部をデータフローネットワークに展開し，作業記憶要素をトークンとして流している．これらの高速化手法は有効であるが，作業記憶の更新が頻繁に起こるような場合には，そのネットワーク中の定数をテストするノードにおいてまだ無駄な処理を行っている．そこでこの無駄な処理を解消するために，マッチング候補の概念を用いたデータフローネットワークによる高速推論アルゴリズムを提案する．このネットワークでは，Reteネットワークでは定数をテストするノードとして同一に扱われているノードをパターン間テストノードとパターン内テストノードに分類して，それらの間にマッチング候補メモリノードを持つことで，無駄な処理を行うことを従来手法に比べて少なくすることができる．また，マッチする作業記憶要素と条件パターンの組を少ない比較回数で特定するために，ID3決定木作成アルゴリズムの考え方を用いたパターン間テストノード作成アルゴリズムを提案する．

A Fast Pattern Matching Algorithm Using Matching Candidates for Production Systems

MITSUNORI MATSUSHITA,[†] MOTOHIDE UMANO,[††,☆] ITSUO HATONO[††] and HIROYUKI TAMURA[††]

Some fast pattern matching algorithms have been proposed to improve the reasoning speed of production systems. In almost all of these methods, rule conditions are represented in a data-flow network and elements in a working memory flow through this network as tokens. These algorithms are effective, however, excessive constant testing still cannot be avoided for instances when the working memory must be frequently updated. This paper proposes a fast pattern matching algorithm for a production system. It uses an improved reasoning network employing matched candidates to circumvent such constant testing which is inherent in conventional networks. Using an example conventional network, the Rete network, we classify constant-test nodes into inter-pattern test nodes and intra-pattern test nodes. Then we introduce memory nodes for matching candidates between these test node classes. This is done in order to find unnecessary matching patterns more quickly. The ID3 algorithm is used to make an efficient inter-pattern test network which is capable of finding patterns in the rule conditions for the working memory element. This algorithm is implemented in Austin Kyoto Common Lisp (AKCL) and is applied to several rule bases with good results.

1. はじめに

エキスパート・システムを作成するツールとしてプロダクション・システムがよく用いられている．しかしプロダクション・システムは，推論に時間がかかるという問題点があった．そこで，様々な高速化の手法が提案されている[1]．その代表的な高速化の手法として，ルールの条件部をReteネットワークと呼ばれるデータフローネットワークに展開し，作業記憶の内容を要素ごとにトークンとして流すReteアルゴリズム[2]がよく知られている．また，Reteアルゴリズムを改良したTREATアルゴリズム[3]や排他Reteアルゴリズム[4]やELITEアルゴリズム[5]なども提案されており，各々のアルゴリズムの有効性が確認されている．

しかし，作業記憶の更新が頻繁に起こるような場合

† NTTコミュニケーション科学研究所
NTT Communication Science Laboratories
†† 大阪大学基礎工学部システム工学科
Department of Systems Engineering, Faculty of Engineering Science, Osaka University
☆ 現在，大阪府立大学総合科学部
Presently with College of Integrated Arts and Sciences, Osaka Prefecture University

には，データフローネットワーク中の定数をテストするノードにおいて，まだ無駄な処理が行われている．

そこで，この無駄な処理を解消するためにマッチング候補の概念を用いたデータフローネットワークによる高速推論アルゴリズムを提案する．

Reteネットワークでは定数をテストするノードとして同一に扱われているノードを，このネットワークでは作業記憶要素とマッチする可能性のある条件パターンを特定するためのパターン間テストノードと，その作業記憶要素と特定された条件パターンが本当にマッチするかを確かめるパターン内テストノードに分類する．そして，パターン間テストノードとパターン内テストノードの間にマッチング候補メモリノードを持つことで，トークンの削除や変更のオーバヘッドを，従来手法に比べて少なくしている．

さらに，パターン間テストノードの構造をID3決定木作成アルゴリズム[6]の考え方を用いて作成する．これにより，少ない比較回数でパターンを特定することが期待できる．

以下，2章では対象としたプロダクション・システムの概要について述べ，3章では従来の高速化手法について簡単に説明し，その問題点について述べる．4章では提案するアルゴリズムについて述べ，5章で研究成果をまとめる．

2. 対象としたプロダクション・システム

本研究は，もともとKyoto Common Lispで記述されているファジィ・プロダクション・システム[7]を高速化することを目標として始めた研究[8]であった．したがって，本論文ではファジィ・プロダクション・システムからファジィ処理に関する機能を取り除いたシステムを用いて高速化アルゴリズムをインプリメントし，実験を行っている．以下にシステムの概要を簡単に説明する．

このシステムでは通常のプロダクション・システムと同様に，ルールベース中のルールと作業記憶を用いて推論を行う．

ルールベースはルールブロック単位に記述する．ルールブロックは複数のルールから構成されていて，各ルールはルール名と確信度（優先度）と条件部と結論部から成る．また，条件部は条件パターンの並びであり，条件パターンにはアトム，変数，リストを記述することができる．変数は先頭が = のアトムである．

また，作業記憶は，作業記憶要素単位に集合で表現する．各作業記憶要素には確信度を設定することができる．

このルールベース中のルールと作業記憶とのパターン・マッチングを行って，発火するルールを特定する．

ルールブロックと作業記憶の例を次に示す．

- ルールブロック

```
(rules  rb1
   (r1 0.8 if   (a  b)
                (c  b)
           then (deposit (a d))
   (r2 0.7 if   (a  d)
                (c =x)
           then (deposit (e =x)) )
```

- 作業記憶

```
(wm wm1 {1/(a d), 0.5/(c b)})
```

ルールブロックrb1にはr1とr2の2つのルールが含まれている．また，作業記憶wm1には(a d)という作業記憶要素が確信度1で，(c b)という作業記憶要素が確信度0.5で記憶されている．このルールブロックrb1と作業記憶wm1を用いて推論を行うとする．ルールr1の条件部の条件パターンのうち，(c b)は作業記憶要素(c b)とマッチするが，(a b)はマッチする作業記憶要素がないため，ルールr1は全体としてマッチしない．また，ルールr2の条件部の条件パターンのうち，(a d)は作業記憶とマッチし，(c =x)は作業記憶要素(c b)とマッチして変数=xにbが代入されるので，ルールr2がマッチすることになる．いまはルールがこの2つしかなく，競合するルールがほかにないのでルールr2が発火し，その結論部（deposit (e =x)）が実行され，作業記憶に作業記憶要素(e b)が付け加えられる．

3. 従来の高速化手法とその問題点

プロダクション・システムは，記述されたルールと作業記憶のマッチングによって推論を進めていくが，ルールや作業記憶要素の数が多くなるにつれて処理時間が多大になってしまい，実用面で問題となる．そこで，プロダクション・システムを高速化するために，様々な研究がなされている．その代表的なものとしてReteアルゴリズムが提案されている[2]．通常のプロダクション・システムは，パターン・マッチング，競合解消，結論部の実行という認知・実行サイクルを繰り返すことで推論を進めていくが，その処理時間の大部分をパターン・マッチングに費やしている．Reteアルゴリズムはそのパターン・マッチングを効率良く実行するためのアルゴリズムである．

3.1 従来の高速化手法

Reteアルゴリズムでは，与えられたルールブロック

中のルールの条件部はReteネットワークと呼ばれるデータフローネットワークに展開され，前回の認知・実行サイクルでのマッチング結果がネットワーク内に保持される．これによって，今回の認知・実行サイクルでの重複するマッチングを省略することができる．

たとえば，次に示すようなルールブロックと作業記憶が与えられているとする．

- ルールブロック

```
(rules (r1 if   (C1 (a1 =x)(a2 4))
                (C2 (a1 =x)(a2 6))
           then (deposit (C4 (a1 8))))
       (r2 if   (C1 (a1 =x)(a2 4))
                (C2 (a1 =x)(a2 7))
                (C3 (a1 6))
           then (deposit (C4 (a1 5)))))
```

- 作業記憶

```
(working-memory
  { (C1 (a1 1)(a2 4)),
    (C2 (a1 1)(a2 6)),
    (C2 (a1 1)(a2 7)),
    (C3 (a1 7))        })
```

2つのルールr1とr2からなるこのルールベースをReteネットワークに展開すると**図1**のようになる．ネットワークには前回の認知・実行サイクルで変更された作業記憶要素だけをトークンとしてルートノードから流し，ネットワーク内の各ノードでマッチングを行う．このとき，前回の推論サイクルにおいて追加された作業記憶要素はプラストークンとして流し，削除された作業記憶要素はマイナストークンとして流す．確信度に変更のあったトークンの場合は，変更前の作業記憶要素をマイナストークンとして流したあとに変更後の作業記憶要素をプラストークンとして流す．

ノードには定数テストノード（constant-test node），メモリノード（memory node），2入力ノード（two-input node），終端ノード（terminal node）の4種類があり，各々次のような働きをする．

(1) 定数テストノード
トークンの属性値の定数が条件を満たすか調べる．

(2) メモリノード
前サイクルで定数テストノードを通過したトークンを保持するもので，2入力ノードの入力部に存在する．2入力ノードと定数テストノードの間に存在するメモリノードを α-mem と呼び，ある2入力ノードの出力部と他の2入力ノードの入力部の間に存在するメモリノードを β-mem と呼ぶ．

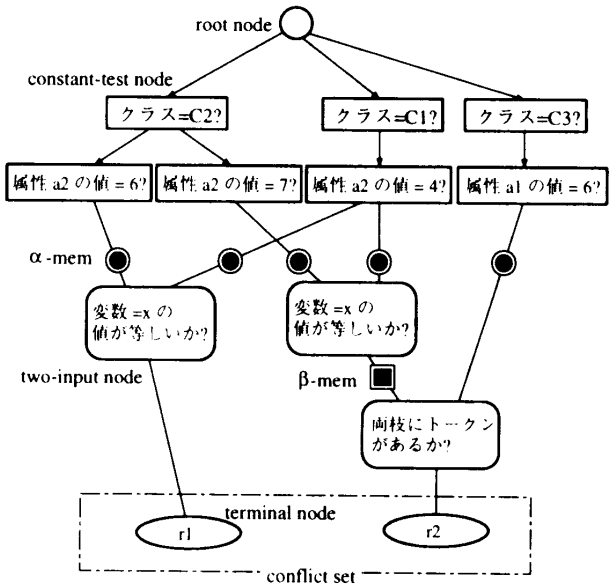

図1 Reteネットワークの例
Fig. 1 Rete network sample.

(3) 2入力ノード
ルールの条件部で共通の変数に同じ定数が代入されているか調べる．

(4) 終端ノード
各ルールを表す．

異なる条件パターンに存在する同じ定数テストノードは共有される．また，2入力ノードでは新しく流れてきたトークンに対し，前サイクルでメモリノードに保持していたトークンとのマッチングを行う．そして変数の値が一致すればメモリノードの内容を適切に変更し，新しくトークンを生成して出力部から流す．

Reteアルゴリズムでは，現在の作業記憶に対するマッチング結果を条件要素単位で保存し，次の認知・実行サイクルでのマッチングに活用する．これにより，変化しなかった作業記憶要素と条件パターン（条件パターンは変化しない）とのマッチングをやり直すという無駄を排除できる．また，ルールの条件部の共通しているパターンを前もって抽出しているため，同じマッチングをまとめて行うことができるので，同一のマッチングを何度も繰り返すという無駄を避けられる．

Reteアルゴリズムの効果は様々なルールで確認され多くのシステムで採用されているが，最も良いアルゴリズムであるというわけではなく，これをもとにした高速推論アルゴリズムもいくつか提案されている．たとえば，作業記憶の更新が頻繁に起こるような場合には，2入力ノードにおけるメモリ変更のオーバヘッドが増加して，速度の低下につながるために，2入力

ノードを動的に作成する TREAT アルゴリズム[3]や，Rete ネットワークの定数テストノード部の排他性に着目して無駄な判定を避ける排他 Rete ネットワーク[4]が提案されており，各々効果をあげている．なお，排他 Rete アルゴリズムの考え方は TREAT アルゴリズムに適用することもできる．このほかの方法については文献 1) を参照されたい．

3.2 従来手法の問題点

次のようなルールを例にして考える．

rule 1 : if P_1 P_2 $\cdots$ P_n then $\cdots$

このルールの条件部をデータフローネットワークで表現すると**図 2** のようになる．

このルールの条件パターン P_i $(i = 1, 2, \ldots, n)$ のほとんどすべての条件がマッチした場合でも，どれか 1 つの条件パターンがマッチしなければこのルールは発火しない．たとえば，図 2 の例で P_1 から P_{n-1} の条件パターンがマッチし，それらを表す α-mem にトークンが到着したとしても，パターン P_n がマッチしない限り，P_1 から P_{n-1} のマッチング，すなわち α-mem に到着するまでに通過する定数テストノードでのマッチングが無駄になる．また，α-mem 以降の 2 入力ノードや β-mem での処理も同様に無駄になる．メモリノードに到着したトークンは，更新されない限り次の認知・実行サイクル以降もそのまま保持されるので，作業記憶の更新がそれほど頻繁に起こらないようなルールではあまり問題にならないが，作業記憶の更新が頻繁に起こるような場合には，通過する定数テストノードの数に比例して更新処理のオーバヘッドが増加する．これは Rete アルゴリズムに限らず，TREAT アルゴリズムや排他 Rete アルゴリズムでも同様である．

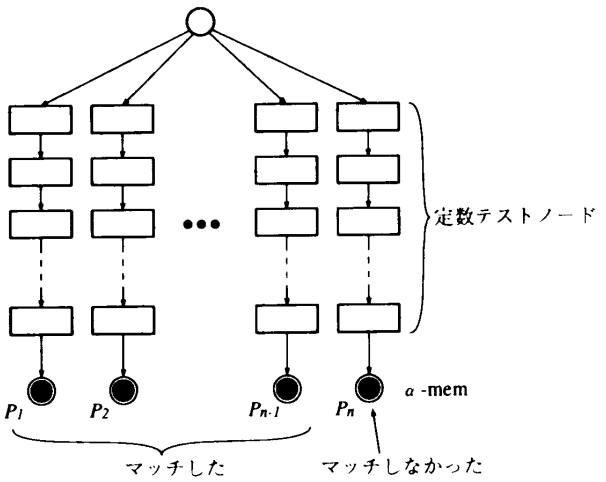

図 2 1 つのルールの条件部のネットワーク表現
Fig. 2 Network expression of the rule conditions.

4. マッチング候補の概念を用いた高速推論アルゴリズム

もし，流されるトークンがどの条件パターンとマッチするかを，より少ない比較回数で効率よく特定することができれば，前節で示したような無駄なパターン・マッチングを回避することができ，推論の高速化が可能となる．そこで，“マッチング候補” という概念を用いた推論ネットワークを提案する．

マッチング候補とは，特定の条件パターンとはマッチするが，それ以外の条件パターンとはマッチする可能性がまったくなくなったトークンを指す．この概念を用いることによって，マッチしない条件パターンを持つルールが容易に分かるので，発火しないことが明らかなルールのパターン・マッチング処理を省略することができる．提案するアルゴリズムの特徴は，従来のデータフローネットワークの定数テストノード部にマッチング候補の概念を持ち込むことで無駄なマッチングを減らすことができるという点にあるので，Rete ネットワークと TREAT ネットワークのどちらにも適用可能であるが，作業記憶の更新が頻繁であることを仮定しているので，本論文では TREAT ネットワークに適用した．

4.1 提案アルゴリズムの概要

例として，次のような 4 項リストの条件パターンを持つルールブロックを考える．

```
(rule  (r1 if (=x b1 c1 d1)
              (a1 b2 =y d2) then …)
       (r2 if (a2 b1 c2 d3)
              (a2 b2 c2 d3) then …)
       (r3 if (a1 b2 =x d2) then …) )
```

ここで，=x，=y は変数で，パターン・マッチングにより値が代入される．このルールブロック中には次の 5 個の条件パターンが含まれている．

p1 : (=x b1 c1 d1)　　p2 : (a1 b2 =y d2)
p3 : (a2 b1 c2 d3)　　p4 : (a2 b2 c2 d3)
p5 : (a1 b2 =x d2)

これらをまとめて条件パターン集合 P と呼ぶことにする．このルールブロックをマッチング候補の概念を用いた高速推論ネットワークで表現すると**図 3** のようになる．このネットワークはパターン間テストノードとマッチング候補メモリノードとパターン内テストノードとパターンノードから構成されている．

このネットワークでは，従来のデータフローネットワークの定数テストノードを，トークンとマッチするパターンを特定するために用いるパターン間テストノー

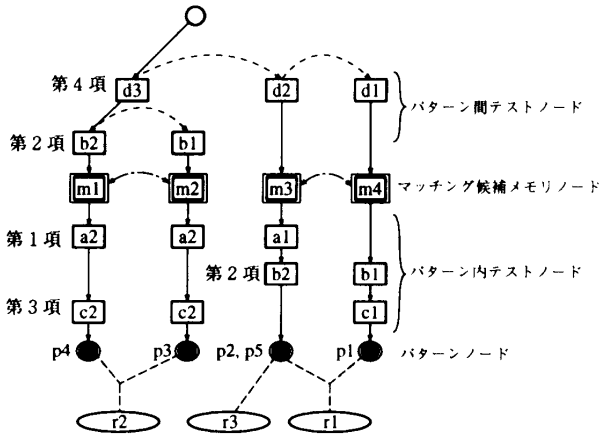

図3 マッチング候補の概念を用いたネットワーク
Fig. 3 Reasoning network using the concept of matching candidates.

ドと，特定された後，残りの属性が本当にマッチするのかを確かめるパターン内テストノードに分類し，パターン間テストノードとパターン内テストノードの間にマッチング候補メモリノードを配置している．ルートノードから流されたトークンは，パターン間テストノードを通過した時点，すなわちメモリノードに到着した時点でどのパターンとマッチする可能性があるか判明している．そのためマッチング候補メモリノードとパターンノードは一意に対応しており，パターン内テストノードには分岐がなく，あるマッチング候補メモリノードにたどり着いたトークンはその下流にあるパターンノードにしかたどり着かない，すなわち，そのパターンノードが表す条件パターン以外のパターンとはマッチしないことが分かる．したがって，マッチング候補メモリノードにトークンが到着していなければ，そのノードと対応しているパターンノードにトークンが到着しないことが分かるので，パターン内テストノードの処理を行わずに，発火しないルールを見つけることができ，無駄なマッチングを減らすことができる．

以下に，各々のノードでどのような処理が行われるかについて述べる．

(1) パターン間テストノード

このノードでは，各条件パターンの持つ特徴的な属性に注目して，トークンがどの属性を持っているかを調べる．特徴的な属性とは，その属性に注目することで，トークンとマッチするパターンを早く特定することができる属性を指す．たとえば，図3の場合には，まずトークンの第4項目に注目し，その属性値がd3であればさらにトークンの第2項目に注目し，その属性値がb2であればパターンp4とマッチする可能性があり，属性値がb1であればパターンp3とマッチする可能性があることが分かる．また，その属性値がd2であればパターンp2，p5とマッチする可能性があり，属性値がd1であればパターンp1とマッチする可能性があることが分かる．

各パターン間テストノードにおいて，トークンがマッチすればその下流のノードにのみトークンを流す．もしマッチしない場合は，排他リンク（図中破線）で結ばれているノードにトークンを流す．流す先がなくなれば，その作業記憶とマッチするパターンはないことが分かる．この排他リンクは排他Reteアルゴリズム[4]の考え方を用いている．

(2) マッチング候補メモリノード

このノードは，パターン間テストノードを通過して到着したトークンを保持するノードである．このノードにトークンが到着しなければ，対応しているパターンノードにもトークンが流れないのでマッチしない．したがってそのパターンを含むルールが発火しないことがこの時点で分かるので，それらのルールに含まれるパターンについてはパターン内テストノードの処理を行う必要がない．

発火しないルールを特定するために，マッチング候補メモリは確認リンクを持つ（図中の一点鎖線）．このリンクはマッチング候補メモリノードを作成する時点で，同じルール内に存在する条件パターンに対応するマッチング候補メモリノードどうしに張る．推論の実行の際には，あるマッチング候補メモリノードに注目して，そのメモリノードにリンクしているメモリノードのうち，1つでもトークンのないものがあれば，そのリンクを張るときに用いたルールは発火しない．したがってこれ以降のマッチングは無駄なのでマッチングを行わず，そのままトークンを保持しておく．すべてのリンクしているノードにトークンがあれば，注目しているメモリノードのトークンはそのまま残し，そのコピーを下流のパターン内テストノードに流す．たとえば，図3で，メモリノードm2，m3，m4にトークンがある場合，m3とm4はリンクしているメモリノードのすべてにトークンがあるので，その下流のパターン内テストノードにトークンを流すが，m2はリンクしているm1にトークン

がないので，その下流のパターン内テストノードにトークンを流さない．

(3) パターン内テストノード
このノードは，そのトークンがパターン間テストノードでテストされなかった属性についてテストするノードである．パターン内テストノードでマッチングに失敗したトークンはマッチング候補メモリノードから削除する．トークンが最終的にパターンノードに到達すればそのトークンとパターンノードが表す条件パターンがマッチしたことになる．たとえば，図3で，メモリノードm4に(a1 b1 c1 d1)というトークンがある場合には，パターン内テストノードを通過し，パターンノードp1に到着するが，(a1 b2 c1 d1)というトークンがある場合には，第2項目がマッチしないのでp1とはマッチしないことが分かるので，そのトークンをマッチング候補メモリノードm4から削除する．

(4) パターンノード
パターンノードは従来のα-memに相当するが，次のサイクルまでトークンを保持しないという点で異なる．それは，提案アルゴリズムは更新処理が頻繁に起こる場合を想定しているからである．すなわち，変更もしくは削除されたトークンの処理のために流されるマイナストークンも，保持しているノードまではプラストークン同様の処理を行う必要があるので，更新処理が頻繁に行われる場合には，マッチング候補メモリノードで前サイクルのトークンを保持する方が，マイナストークンの処理を早く行うことができるからである．

推論は，従来の推論方法と同様に作業記憶要素をトークンとしてルートノードから流すことで開始する．流されたトークンはパターン間テストノードによって，マッチする可能性のあるパターンに対応したマッチング候補メモリノードに到着する．すべてのトークンがマッチング候補メモリノードに到着した時点で，メモリノード間の確認リンクを用いてパターン内テストノード以降の処理を行うトークンを限定する．これによって限定されたトークンは，パターン内テストノードへ流される．

パターン内テストノードを通過したトークンはトークンの処理順序に応じて動的に生成された2入力ノードで変数の一貫性チェックを行われる．この処理以降はTREATアルゴリズムと同様である．なお，パターンノードのトークンは次の認知・実行サイクルに移る前に消去する．こうすることで次サイクルのマイナストークンの処理にともなう効率の低下を防ぐことができる．

このアルゴリズムでは，パターン間テストノードでのマッチング回数が少ないほど，効率的な推論を行うことができる．しかし，パターンの属性のうち，どの属性から調べていけば効率の良いパターン間テストノードを作成できるかは一概にはいえず，条件パターンの形式に依存する．そこで，効果的なパターン間テストノードを作成する手法として，ID3決定木作成アルゴリズムを応用したパターン間テストノード作成手法を提案する．

4.2 ID3の考え方を応用したパターン間テストノードの作成

ID3決定木作成アルゴリズム[6]は，「データを分類するときのテスト回数の期待値を最小にする」ことを目的としたアルゴリズムである．そのために獲得情報量（相互情報量）の期待値を最大にする属性を決定木のテストノードとして選択している．

ℓ個の属性$A_1, A_2, \ldots, A_\ell$があり，各々$A_i = \{a_{i1}, a_{i2}, \ldots, a_{im}\}$（$m$は$A_i$によって異なる）の属性値を持ち，分類クラスは$C = \{C_1, C_2, \ldots, C_n\}$とする．このような属性と分類クラスを持つ学習データの集合があるときに，ID3により決定木を作成するアルゴリズムは次のようになる．

1. すべての学習データを対応付けたノード（ルートノード）を生成する．
2. ノードに対応付けられた学習データ集合Dがすべて同じ分類クラスC_kに属するならば，そのノードを葉ノードとし，分類クラスC_kをラベル付けする．
3. そうでなければ，そのノードは葉ノードではないので，次の処理を行う．
   (a) 各属性A_iに対して，その属性値によりDを部分集合に分割し，獲得情報量$G(A_i, D)$を下の式で計算する．
   (b) 計算した獲得情報量が最大となる属性をこのノードのテスト属性A_{max}とする．
   (c) テスト属性A_{max}の各部分集合に対して，もとのノードと各部分集合のノードとを結ぶ枝に，対応する属性値$a_{max,j}$をラベル付けする．
   (d) 作成したすべてのノードに対して，2.から実行する．

ここで，獲得情報量は次の式で計算する．このとき，D_{C_k}を分類クラスがC_kであるDの部分集合とし，

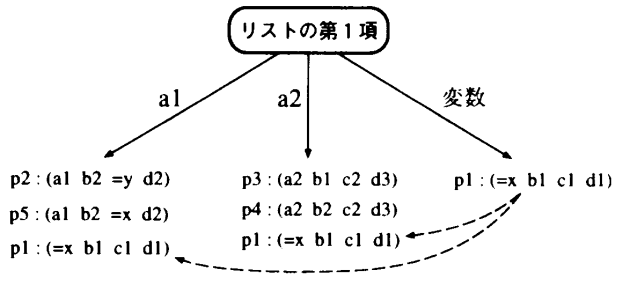

図4 第1項目での分類
Fig. 4 First attribute classification of a pattern list.

$D_{a_{ij}}$ を属性 A_i の値が a_{ij} である D の部分集合とする．

$$G(A_i, D) = I(D) - E(A_i, D) \tag{1}$$

ただし，

$$I(D) = -\sum_{k=1}^{n} \left(\frac{|D_{C_k}|}{|D|} \cdot \log_2 \frac{|D_{C_k}|}{|D|} \right) \tag{2}$$

$$E(A_i, D) = \sum_{j=1}^{m} \left(\frac{|D_{a_{ij}}|}{|D|} \cdot I(D_{ij}) \right) \tag{3}$$

ここで，$|D|$ は D の要素数である．

アルゴリズムの3において計算される獲得情報量 $G(A_i, D)$ は負になることはなく，属性 A_i で分類しても変化がないときに0となり，それ以外は正の値をとる．また，情報量 $I(D)$ には最大値 $\log n$ が存在し，それは確率がすべて等しいときである．

この考え方を，実際のルールの条件パターンからパターン間テストノードを作成するときに用いることで，効率的なパターン間テストノードを作成することが期待できる．

例として前節で用いたルールブロックの条件パターン集合 P を用いる．まず，これらを分類することを考える．分類する基準としてはリストの第1項目から第4項目のシンボルアトムが候補としてあげられる．

P の条件パターンを仮にリストの第1項目で分類したとする．分類する項目はa1とa2と変数である．変数は表記が異なっていてもマッチングに関しては同様の振舞いをするため，まとめて“変数”という属性値であると見なす．するとパターンp1の第1項目は変数なので，データとして属性値a1，a2が与えられた場合でも，マッチすることになる．そこで，排他性を保証するためにp1を他のノード（ここではa1とa2のノード）にコピーする．この処理によって，変数の枝は属性値a1，a2以外の属性値であると見なせる．このように分類することによって**図4**のような部分木ができる．このように変数属性のパターンを他の枝にコピーするために，見かけ上のパターン数は増加する．変数の数はリストの各項で異なるので，選択する項によって，見かけのパターン数は異なることになる．

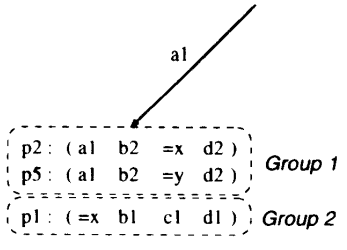

図5 属性値a1の枝でのクラス分け
Fig. 5 Branch classification of attribute value a1.

そして，リストの第1項目で分類したノードでの情報量を計算する．例としてa1の場合を考える．各パターンにはID3で用いるデータとは異なり，クラス名が与えられていない．そこで，**図5**に示すように形式が同一のパターンを同じクラスに分類して仮のクラス名（ここではGroup1，Group2）を与え，そのクラスをもとに情報量を求める．形式が同一のパターンとは，リストの長さが等しく，かつその構成要素が等しいパターンのことである．ただし変数は同一の要素として扱う．属性値a1を選んだときのパターンのノードの情報量 $I(P_{a1})$ は，次のように求められる．

$$\begin{aligned} I(P_{a1}) &= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \\ &= 0.918 \text{ (bit)} \end{aligned}$$

同様にして属性値a2と変数を選んだときの各々のノードでの情報量 $I(P_{a2})$，$I(P_{変数})$ を計算すると次のようになる．

$I(P_{a2}) = 1.585$ (bit)，$I(P_{変数}) = 0.0$ (bit)

したがってリストの第1項目で分類した場合の情報量の期待値 E(第1項, P) は次のようになる．

$$\begin{aligned} E(\text{第1項}, P) &= \frac{3}{7} \times 0.918 + \frac{3}{7} \times 1.585 + \frac{1}{7} \times 0.0 \\ &= 1.073 \text{ (bit)} \end{aligned}$$

同様に，リストの第2項目，第3項目，第4項目で分類した場合の情報量の期待値は各々次のようになる．

E(第2項, P) $= 0.951$ (bit)

E(第3項, P) $= 0.973$ (bit)

E(第4項, P) $= 0.400$ (bit)

情報量が最小になるのはリストの第4項目で分類した場合なので，ここではそれを用いて分類を行う．

以上の操作を，1つの枝にあるクラスが1つになるまで再帰的に繰り返すことで，最終的に**図6**のような決定木が得られる．決定木の葉ノードがマッチング候

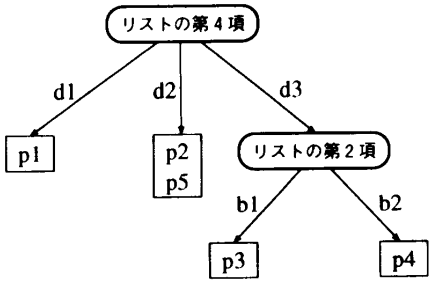

図6 パターンによる決定木
Fig. 6 Decision tree from a pattern.

補メモリノードに相当する．この決定木から図3のパターン間テストノードのように，排他性を持つパターン間テストノードが得られる．このとき，同じルールに含まれる条件のマッチング候補メモリノードどうしにリンクを張る．排他リンクを持つノードでのマッチングでは，変数属性を除いてその下流に流れるパターン数が多いものから順に調べ，最後に変数属性に無条件にトークンを流すようにする．パターン内テストノードで調べる属性は，各葉ノードでその葉ノードに到着するまでに通過するテストノードにおける属性以外のものである．

ここでは，すべての条件パターンが同じリスト長であったが，パターンのリスト長が異なるときは，まずリスト長で分類してこのアルゴリズムを適用する．リスト内に含まれる要素がリストの場合は，アトムを長さ0のリストと考えて，同様にリスト長で分類すればよい．

4.3 性能評価

インタプリタのみの場合と排他 TREAT アルゴリズムを用いた場合と提案アルゴリズムを用いた場合の各々の実行速度を，次の4つのルールベースで比較した．排他 TREAT アルゴリズムと比較する理由は，作業記憶の更新が頻繁であることを仮定し，パターン間テストノードに排他性を採用しているからである．

rb1：15ルール，15パターン
「風が吹けば桶屋が儲かる」という小噺をルールにしたものである．このルールベースによる推論では，作業記憶要素の更新がほぼ100%であるため，履歴情報を用いることができない．

rb2：15ルール，45パターン
履歴情報を用いることができる場合の効果を調べるためにrb1の各ルールの条件部に，つねに成り立つような条件パターンを追加したものである．たとえばrb1に含まれているルール

```
(r1  if   (blow wind)
     then (deposit (flit dust))
          (delete (blow wind) 1))
```

の条件部に (foo (bar)) と (bar (foo) bar) という余分な条件パターンを追加して

```
(r1  if   (foo (bar))
          (bar (foo) bar)
          (blow wind)
     then (deposit (flit dust))
          (delete (blow wind) 1))
```

というルールに変更し，作業記憶に (foo (bar)) と (bar (foo) bar) の2つの作業記憶要素を追加した．これにより，作業記憶の更新が3割ほどになる．

rb3：25ルール，45パターン
被験者の問診結果から，どのような種類のストレスを感じやすい体質かを診断するルールベースである．

rb4：18ルール，36パターン（オーバヘッド評価用）
ルールの条件部に共通する条件がなく，また履歴情報も用いることができないようなルールでデータフローネットワークに展開しても高速化できないようなルールベースである．これを用いてオーバヘッドを評価する．

これらのルールベースを用いて行った実行結果を図7に示す．

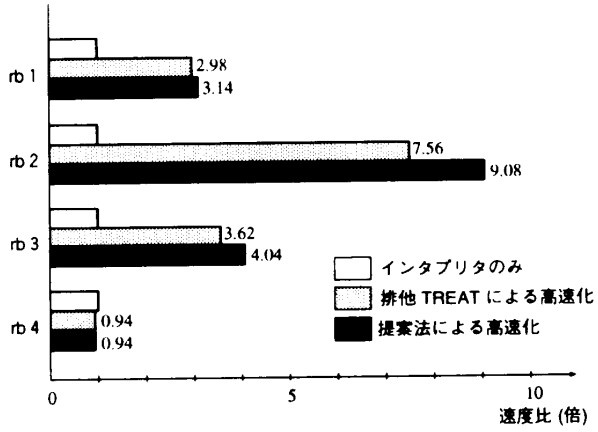

図7 性能比較
Fig. 7 Performance comparisons.

ルールベース rb1，rb2，rb3 の結果から，提案アルゴリズムを用いることで，排他 TREAT アルゴリズムに比べて，5%から20%程度の高速化が達成されていることが分かった．また，ルールベース rb4 から，ネットワーク自体のオーバヘッドも排他 TREAT アルゴリズムと同程度であることが分かる．したがって，マッチング候補の概念を導入した提案アルゴリズムは有効であるといえる．

5. おわりに

本研究では，従来のプロダクション・システムの高速化手法において用いられているアルゴリズムの無駄を指摘し，その無駄を回避するために，マッチング候補の概念を用いた推論ネットワークを用いた高速推論アルゴリズムを提案した．このアルゴリズムを用いることによって，更新が頻繁に起こるような場合の推論をより高速に行うことができるようになった．また，このネットワークの効果的なパターン間テストノードを作成するためにID3決定木作成アルゴリズムの考え方を用いた．

今後，より多くの実際的なルールベースを対象にアルゴリズムの有効性を調べる必要がある．

また，作業記憶や条件パターン中にファジィ値が記述された場合について考察し，ファジィ・プロダクション・システムの高速化を図る予定である．

なお，提案した推論ネットワークはSUN Microsystems社製のSPARC station上のAKCL（Austin Kyoto Common Lisp）上でインプリメントした．

参考文献

1) 増位庄一，田野俊一：プロダクションシステムの高速化，人工知能学会誌，Vol.6, No.1, pp.38–46 (1991).
2) Forgy, C.L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol.19, No.1, pp.17–37 (1982).
3) Miranker, D.P.: TREAT: A Better Match Algorithm for AI Production Systems, *AAAI-87*, pp.42–47 (1987).
4) 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, pp.768–775 (1987).
5) 田野俊一，増位庄一，舩橋誠壽：ELITEアルゴリズム：確率コストモデルに基づくジョイントネットの最適化方式，情報処理学会論文誌，Vol.35, No.5, pp.725–738 (1994).
6) Quinlan, J.R.: Discovering Rules by Induction from Large Collections of Examples, *Expert Systems in the Micro Electronics Age*, Michie, D.(Ed.), Edinburgh University Press (1979).
7) Umano, M.: Implementation of Fuzzy Production System, *Third International Fuzzy Systems Association Congress*, Seattle, Washington, USA, pp.450–453 (1989).
8) 馬野元秀，松下光範，鳩野逸生，田村坦之：ファジィ・プロダクション・システムの高速化技法，第21回知能システムシンポジウム，pp.205–210 (1995).

(平成7年11月9日受付)
(平成8年4月12日採録)

松下 光範（正会員）

1995年，大阪大学大学院基礎工学研究科物理系専攻制御工学分野前期課程修了．同年4月日本電信電話（株）入社，現在に至る．大規模知識ベースに関する研究に従事．本研究は大学院在学時に行ったものである．

馬野 元秀（正会員）

1979年，大阪大学大学院基礎工学研究科物理系専攻情報工学分野後期課程修了．同年4月岡山理科大学理学部応用数学科講師．1985年4月大阪大学大型計算機センター助手，その後同センター講師，助教授を経て，1991年1月同大学工学部精密工学教室助教授，1993年10月同大学基礎工学部システム工学科助教授，1996年4月より大阪府立大学総合科学部教授，現在に至る．ファジィ集合論の応用，特に，プログラミング言語，データベース，知識情報処理への応用に関する研究に従事（工学博士）．

鳩野 逸生（正会員）

1986年，大阪大学大学院工学研究科精密工学専攻博士前期課程修了．同年4月日本電気（株）C&Cシステム研究所勤務．1988年8月より大阪大学工学部精密工学教室助手．1993年10月より同大学基礎工学部システム工学科助手，1996年4月より同講師，現在に至る．生産スケジューリングおよび離散事象システムのモデリングに関する研究に従事．1992年システム制御情報学会椹木記念奨励賞受賞（工学博士）．

田村 坦之

1962 年大阪大学工学部精密工学科卒業．1964 年同大学院工学研究科原子核工学専攻修士課程修了．同年 4 月三菱電機（株）中央研究所勤務．1971 年 10 月大阪大学工学部精密工学教室助教授，1987 年 8 月同教授．1993 年 4 月同大学基礎工学部システム工学科教授，現在に至る．その間，1966～68 年米国スタンフォード大学大学院 Engineering-Economic Systems 学科に留学，1972～73 年英国ケンブリッジ大学客員研究員，1986～88 年統計数理研究所併任．統合システム解析の方法論とその生産システム・公共システムへの応用に関する研究に従事．1976 年計測自動制御学会論文賞，1990 年システム制御情報学会椹木記念賞論文賞，1991 年日本 OR 学会事例研究奨励賞受賞（工学博士）．