人 口 知 能 84－1
（1992. 9. 10）

変数表現と否定条件のためのReteアルゴリズムの拡張

高野 啓、吉良賢治、澤本 潤
三菱電機(株) 情報電子研究所

プロダクションシステムの高速実行手法であるReteアルゴリズムは、元来OPS言語を対象に開発され、その記述力は暗にOPS言語の言語仕様に制約を受けている。ゆえにReteを実装する処理系では、ルールの条件部の言語仕様にはOPS言語に由来する構文上の制約を受ける。

ここで「構文上の制約」とは、(1) 変数の使用方法、(2) 否定条件の記述、に関するものである。前者は、「WAEによらない条件で使用される変数は、先行する条件パタン中に現れたものに限る。」というもので、そのような条件中でだけ現れる変数の使用が不可になる。後者は、「連言を否定する手段がない。」というものである。

本論文では、上記の制約と、それらを解消する方法を述べる。

An Extension of Rete Algorithm for Variable Expressions and Negative Conditions.

Akira Takano, Kenji Kira, Jun Sawamoto

Computer & Information Systems Laboratory, Mitsubishi Electric Corp.

The Rete algorithm was originally developed for OPS language, and Rete's describability is implicitly restricted by OPS's language specification. Therefore Rete-based Production system interpreters also inherit OPS's syntactic restrictions.

'Rete's syntactic restriction' proposed in this paper concerns 1)Variable usage,2)Negative conditions. The former means "The variables appear in a condition which doesn't depend a WAE should appear in a preceding pattern." So a variable which appears only in such conditions are prohibited. The latter means "There is no way to negate and-conjunction".

This paper describes the details of those restrictions and the method to reduce them.

1. はじめに

Rete アルゴリズム[1] は、プロダクションシステムの高速実行手法として、多くのプロダクションシステム処理系（以下 PS）に適用され、PS の標準的アルゴリズムとしての地位を占めているといえる。また、Rete を基盤として、さらに高速化するアルゴリズム[2][3] 、並列処理計算機への適用[4] 、Rete の問題点を明らかにし改善したアルゴリズム[5][6] などが提案されている。

これらの提案は、すべて Rete を上回る高速化を目的としている。

ところで、Reteは元来OPS言語に実装されたものであり、Reteで扱える文法はOPS言語の制約を受けているが、これまでにこの問題に言及した報告はされていない。

本論文では、まず Rete アルゴリズムが持つ文法的制約として、Rete での変数表現と否定条件の表現について述べる。続いて、それぞれの制約を解消する手法を述べる。なお、本論文中の用語およびルールの文法は OPS5[7] に準ずる。

2. Rete アルゴリズムの文法的制約

2.1. Rete の変数表現

ReteによるPSでは、プロダクションルール中の各変数の位置を、Reteネット中のノード関数の引数として保持する。これによって、パタン照合で各ノード関数を評価する時に、各トークンから変数値を直接得る。

OPS5 を例に Rete の変数表現とその使用方法について述べる。図 1 のルールの各変数<x>((1)～(3)) は、OPS5 の Rete ネット中で図中に示した各インデクスに対応する。それぞれのインデクスは、次のようにして番号づけている。

1) 1 入力ノード変数テストの場合：(<テスト名> <パタン中初出位置> <比較位置>)
2) 2 入力ノード変数テストの場合：(<テスト名> <左辺中初出位置> <比較位置>)

<左辺中初出位置>は、テストを実施する変数がルール左辺のどこで始めて使われたかを示す値であり、パタン番号とパタン内の変数番号を合成して作られる。(変数<x>がN番めのパタンのM番めの変数として現れた時、N*10000+M を<左辺中初出位置>とする。)<パタン中初出位置>、<比較位置>は、パタン中の各項目位置を左から順に番号づけたものである。

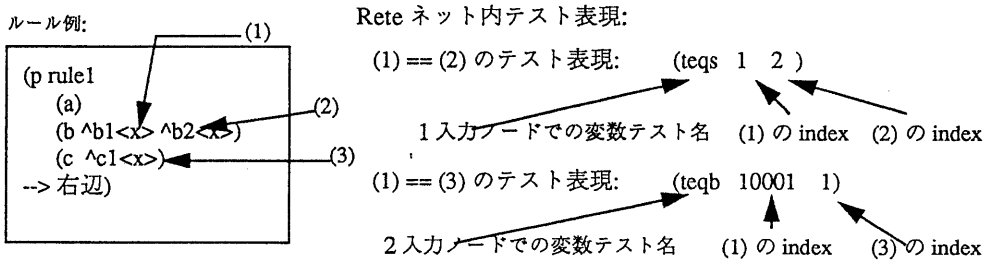

図 1. OPS5 における変数表現

ネットワーク中に記述された関数群は、Rete ネットにトークンが追加／削除された時に評価される。その際、ルール中の変数の評価にかかる部分は、関数中のインデクスをもとに、トークン(2 入力ノードでは、いくつかのトークンが結合された「合成トークン」になる。) 中の変数にあたる位置を検出して、その位置の値を用いて評価される。例えば、トークン (b 20 40) が Rete に追加され、図1の(teqs 1 2)を評価する際には、インデクス '1'の値としてトークン中の1番めの(0オリジン)要素である'20'を用いる。合成トークン((a)(b ^b1 20 ^b2 40))があり、図1の(teqb 10001 1)を評価する際には、インデクス '10001'の値として、1番めの(0オリジン) トークン中の1番めの(0オリ

ジン) 要素である'20'を用いる。

以上のように、すべての変数はインデクスとして表現され、その値はトークンから得る。したがって、すべての変数は条件パタンの中で最初に現れる必要がある。このような変数管理方法は、パタンとして表されない条件（「手続き的条件」と呼ぶ。）の記述力に影響する。具体的には、手続き的条件にだけ現れる変数（「手続き変数」と呼ぶ。）が使えない、といったことがおこる。ここで、便宜的に手続き的条件を':'で始まるものとし、次のルールを例に考える。

```
(p rule1
   (a ^a1 <x>)
   : (get-value <y>))
--> ...)
```

'get-value' は、引数に値を代入する関数であるとする。<y>は手続き変数である。変数<y>には、関数 get-value により何らかの値が設定されるはずだが、その値を右辺に伝える手段がない。なぜなら、手続き的条件は条件パタンとは異なり、対応するトークンを持たないので、<y>の変数情報をトークンとして送れないからである。したがって、Rete では、手続き変数は使用できない。

2.2. Rete による否定条件の表現

Rete ネットワーク内で、ルール左辺中の否定条件が結合する位置の２入力ノードは、通常の条件が結合する２入力ノードと動作がことなる。まず、通常の２入力ノードの動作は、次のようである。

1) +トークンの到達時には、反対側のメモリ中の各トークンと照合を行ない、成功したものとの組合せを β メモリに格納し、+トークンとして次のノードへ流す。
2) -トークンの到達時は、反対側のメモリ中の各トークンと照合を行ない、成功したものは、その組合せを β メモリから除き、-トークンとして次のノードへ流す。

これに対し、否定条件にかかる２入力ノードの動作は、次のようになる。

1) 左側（ルール中否定条件の上側）からの＋トークン到達時には、右側の各トークンと照合し、すべて失敗した場合、β メモリに格納して、次ノードへ流す。成功した組合せがある場合は、その数をトークンに記録する。
2) 右側（否定条件を示すリンク）からの＋トークン到達時には、左側の各トークンと照合し、すべて失敗した場合には、何もしない。照合に成功した左側トークンで、他の右側トークンと照合に成功していなかったものは、これを－トークンとし、β メモリから除き、次ノードへ流す。成功していたものは、その数をインクリメントする。
3) 左側からの-トークン到達時には、これが β メモリにあれば除き、次のノードへ送る。
4) 右側からの-トークン到達時には、左側の各トークンとの照合において、成功したものの数をデクリメントする。その結果、0 になった左側トークンは＋トークンとして β メモリに格納し、次ノードへ送る。

これらの中で、右側のトークンは、ルール中の単独のパタンであることが想定されている。これは、OPS5 の否定条件は、単一パタンの否定しか扱わないことによるものと言えるが、したがって、Rete では、「肯定条件の連言を否定する」ような条件記述ができない。

以上が、否定条件部分の Rete の動作概要と、そこに生じる制約であるが、否定条件に関しては、他にも２つの制約がある。ここにまとめておく。

1) 否定条件を左辺の先頭に書けない。
   Rete の否定条件は、照合テストの対象として、左辺中で否定条件よりも前に書かれている条件を扱う。前述の動作では、先頭に否定条件を記述した場合の動作が説明できず、OPS5 では、

禁止されている。

2) 否定条件中の変数の扱い

前述のように、照合の対象は、その否定条件に先行する条件群に限られるので、否定条件中の変数名は、先行するパタンの中ですでにその変数が現れている場合にだけ意味がある。

```
(p rule1 (...) (a ^a1 <x>) -(b ^b1 <x>) --> ...)---------- 制約あり (1)
(p rule2 (...) -(b ^b1 <x>) (a ^a1 <x>) --> ...)---------- 制約なし (2)
(p rule3 (...) -(b ^b1 <y>) (a ^a1 <x>) --> ...)---------- 制約なし (3)
```

図2. 否定条件中の変数

例えば、図2の各ルールで、変数<x>が先行パタンでは現れないとした時、図2(1)のルールと、(2)のルールでは動作が異なり、(1)では、(a ^a1 <x>)でマッチした<x>の値は(b ^b1 <x>)のマッチに影響するのに対して、(2)で(b ^b1 <x>)でマッチした<x>は(a ^a1 <x>)には影響しないか、構文エラーとなる。影響しない場合、(2)のルールは、意味的には(3)のように、無関係の変数を書いた場合と同等になる。したがって、否定条件中の変数と他のパタン中の変数で照合を行なうには、対象のパタンを否定条件よりも先に書く必要がある。

3. Reteの変数表現を拡張する

3.1. 拡張方法

2.1.で述べた変数表現の制約を解消する方法について述べる。ここで述べる手順により、ルール左辺の手続き的条件において、手続き変数の使用が可能になる。

2.1.で述べた制約は、変数値をトークンから直接に得ることから生じている。著者は、変数の各情報は「変数テーブル」としてトークンから分離することとし、パタンマッチの際には、このテーブルを操作することで、変数情報を得ることとした。以下に詳細を述べる。

1) 1入力ノードでの動作

Reteネットに新たに追加されたトークンは、初期（Reteネットに投入された時）にサイズ0の変数テーブルを持つ。各1入力ノードの新変数値の出現にあたるノード(図3(1))では、テーブルのサイズを拡張し、変数値を格納する。パタン内の変数比較にあたる位置(図3(2))では、テーブル中の位置(図3の場合、1と2)を用いて、値比較を行なう。

これらのようにして1入力ノードでの照合に成功した時、メモリノードに到達したトークンは、パタン内に記述された変数の数と同じサイズの変数テーブルを持つことになる。

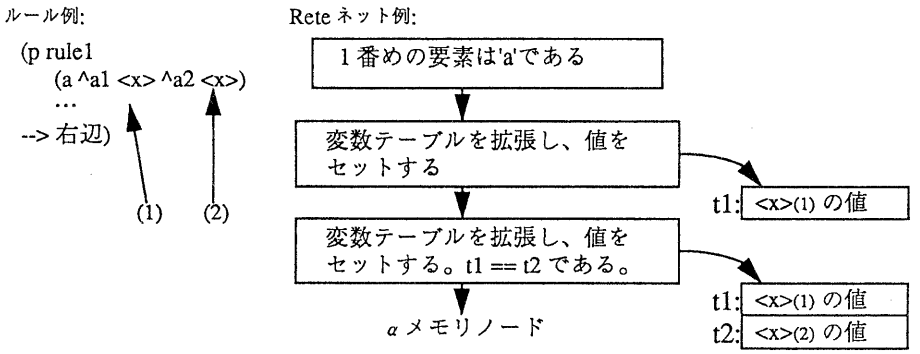

図3. 1入力ノードでの変数テーブル

2) 2入力ノードでの動作

パタン同士が結合する2入力ノードでは、図4のように、左側トークンの変数テーブルと

右側トークンの変数テーブルの所定位置を比較する。照合に成功した時は、トークンを合成するとともに、左右の変数テーブルも結合し、βメモリに格納する。

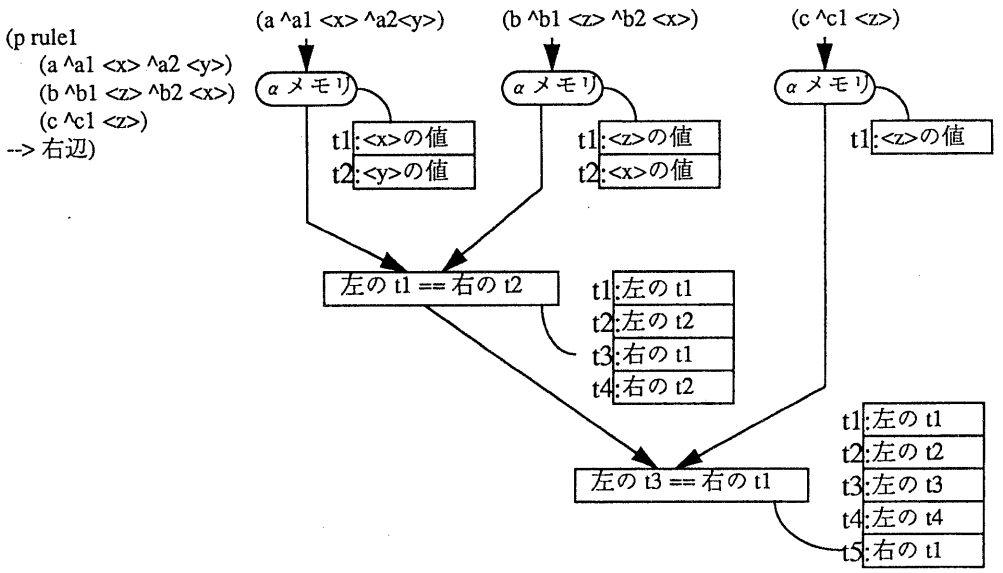

図 4. 2 入力ノードでの変数テーブル

ここで、手続き的条件がこの 2 入力ノードに付随する場合、βメモリの格納前にその評価を実施し、手続き条件中の新変数の数だけ変数テーブルを拡大し、値を格納する。
図 5 に手続き変数を使用するルールの例を示す。図中の関数 get-value は、その引数に値を代入する関数であるとする。このような関数を扱う 2 入力ノードでは、新変数のために変数テーブルを拡張し、get-value の結果をその拡張箇所に格納する。後続の条件では、変数テーブルの内容を参照できるので、手続き変数への代入結果を使用できる。

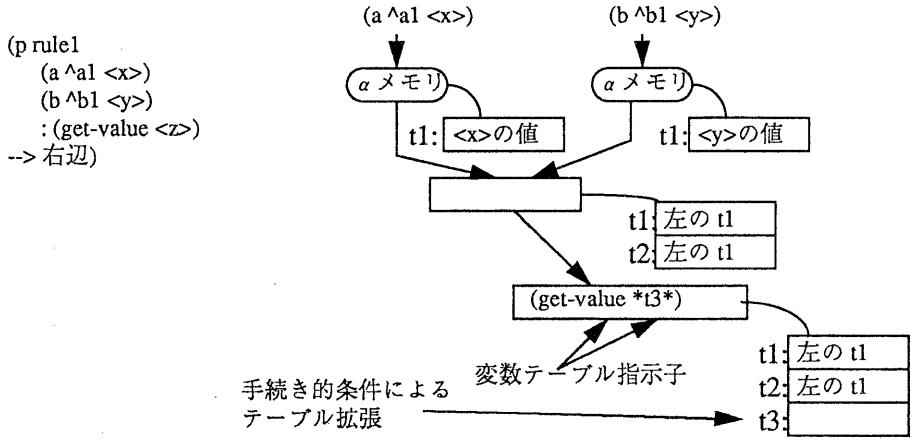

図 5. 手続き的条件での変数テーブル

3) ルールコンパイル時の処理
1)、2) のような動作のために、Rete ネット作成時に、ルール中の各変数の左辺全体での出現順

と、パタン内での出現順を解析する。この解析結果から、Rete ネットでの照合関数を作成する。(図 6)

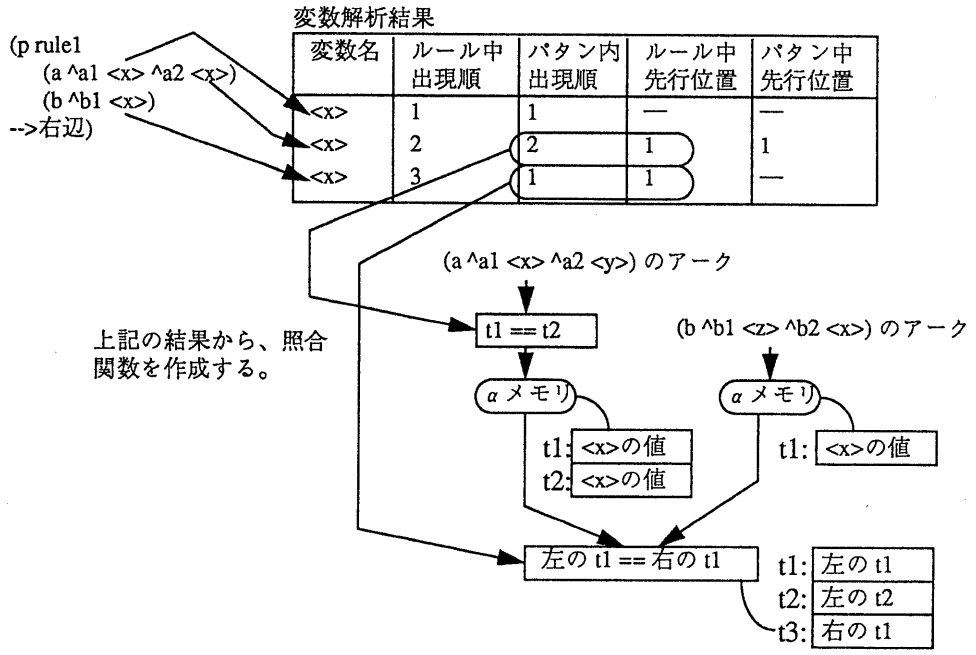

図 6. コンパイル時の処理

以上、変数表現の拡張方法を述べた。

3.2. 考察

著者の提案する方法により、前述のように手続き的条件中での変数の記述性を強化できるが、Rete と比べて、ノード共有率が低下する問題が生じる。

元来、Rete では、最初に出現する変数に当たるところでは何の操作も行なわず、Rete 上で 1 入力ノードを割り当てないが、本案では、変数テーブルへの値格納が必要であり、1 入力ノードを割り当てる。従って、例えば、パタン'(a ^a1 <x>)'と'(a ^a1 <x> ^a2 <y>)'のような組合せは、Rete ではメモリノードの直上までが共有できたのに対し、本案では'<x>'に当たるノードまでが共有の対象であり、しかもノード数は増加する。

この点に関し、著者は、ノード共有の有無が推論効率に及ぼす影響を大と考えず、(この点 Miranker[5] と同様) 本案による記述性の拡大を優先するものである。

4. Rete の否定表現を拡張する

4.1. 拡張方法

否定条件中に複数のパタンを記述した条件記述 (これを「連言否定」と呼ぶ) を可能にするため、まず、連言否定の Rete 向けの解釈方法について考察し、続いて、著者が採用した解釈方法に基づいて Rete アルゴリズムを拡張する方法を述べる。

著者は、連言否定の解釈を、次のような論理式にしたがって行なうことにした。ここで、連言否定に先行する条件群を「先行パタン」としてまとめ、連言否定内部の各パタンを「パタン 1」～「パタン N」で表す。

(先行パタン∧ ¬パタン1)∨(先行パタン∧ パタン1 ∧ ¬パタン2)∨…
∨ (先行パタン∧… ∧¬パタンN)

この論理式は、偽であるパタンiを発見するまで、各パタンを評価することを意味する。この解釈にしたがって拡張したReteネットワークの概要を図7に示す。以下、図7のネットワークの動作を述べる。図7で、太線のリンクは、従来の2入力ノード間のリンク（「通常リンク」と呼ぶ）とは別にここで新たに追加するものである。これを「否定リンク」と呼ぶ。

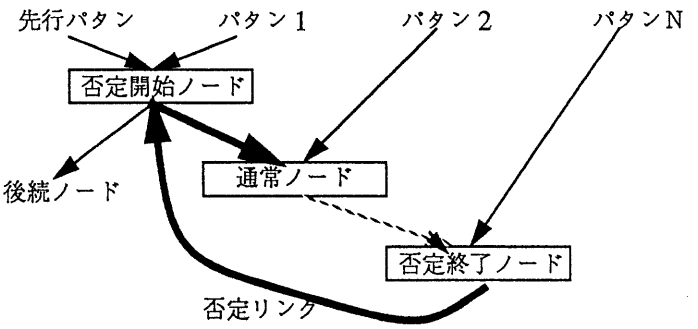

図7. 拡張したReteネット表現

否定リンクの導入にともない、2入力ノードを次の3種類に分類する。

1) 否定開始ノード
「通常リンク」「否定リンク」の両方を持つ2入力ノードで、連言否定が始まる位置を示す。
2) 否定終了ノード
「否定リンク」だけを持つ2入力ノードで、連言否定が終る位置を示す。
3) 通常ノード
「通常リンク」だけを持つ2入力ノードである。連言否定の内部に置く。

上記のうち、「通常ノード」の動作は、従来の2入力ノードを同様である。
以下、「否定開始ノード」「否定終了ノード」の動作を解説する。

1) 否定開始ノードの動作

i. ＋トークンが、左側（先行パタン）から来た時
＋トークンにID番号を付加し、後続ノードへ合成トークンとして送る。また、右側の各トークンとのパタンマッチを行ない、成功した組合せの合成トークンを否定リンク側のノードへ送る。否定リンク側に続くノードでは、通常の2入力ノードと同様の動作になるので、パタンマッチに成功する限り、前述の論理式のように偽となるノードに到達するまで評価される。

ii. ＋トークンが、右側から来た時
左側の各トークンとのパタンマッチで、成功した時は合成トークンを否定リンクに送り、前述と同様連言否定内部でのパタンマッチを行なう。左側にあったトークンは、すでに後続ノードへ送られている。

iii. -トークンが、左側から来た時
引続き後続ノードへ-トークンを送るとともに、否定リンク側に合成トークンがある場合には、引続き否定リンク側にも-トークンを送る。

iv. -トークンが、右側から来た時
(iii) と同様、否定リンク側に-トークンを送る。

2) 否定終了ノード

i. ＋トークンが来た時
否定終了ノードにおいて、左右どちらかの方向から＋トークンが来て、パタンマッチに成功した時は、連言否定内のすべての条件項が真になったことを意味する。このとき、パタンマッチの対象になっている左側トークンは、前述の 1)iii) で付加した ID 番号を持つ。この ID 番号をキーとして、否定リンクを経由して否定開始ノード中にある同一 ID 番号のトークンを-トークンとして継続ノードへ送る。これにより、連言否定が真から偽に転じたことを表す。
なお、否定終了ノードでは、従来の Rete と同様、左側の各トークンに対しパタンマッチに成功している右側トークンの数を記録する。この数が正の時に右側からのトークンとのパタンマッチに成功した時は、-トークンは送らず、単にマッチした数をインクリメントする。

ii. -トークンが左側から来た時
β メモリ中にこれに該当する合成トークンがあれば、これを削除する。さらに、否定リンクが示す否定開始ノードに、この-トークンと同じ ID 番号を持つトークンがある場合は、-トークンは連言否定内部で生じた（連言否定中のどこかのパタンが偽になり、したがって全体が真に転じた）ことを意味する。このときは、その-トークンと同じ ID 番号を持つトークンを否定リンクが示す否定開始ノードから後続ノードへ送る。これにより、ルール中の連言否定以降のパタンとのマッチを行なう。

iii. -トークンが右側から来た時
その-トークンにマッチしていた左側トークンがあり、その左側トークンにマッチしていた右側トークンは他にない場合は、連言否定が偽から真に転じたことを意味する。この時は、その左側トークンと同じ ID 番号を持つトークンを、ii) と同様に対応する否定開始ノードの後続ノードへ送る。

以上のようにして、Rete アルゴリズムの拡張として連言否定の解釈実行を行なう。

4.2. 考察

ここでは、前述の方法を採用した理由について述べる。連言否定の解釈としては、式としては「先行パタン ∧ ¬ (パタン 1 ∧ … ∧ パタン N)」のように考えて、連言否定の部分を副木とした (図 8.)Rete ネットを構築することが考えられる。しかし、この方法には、4.1.と比較して 2 つの問題点がある。以下、順に述べる。

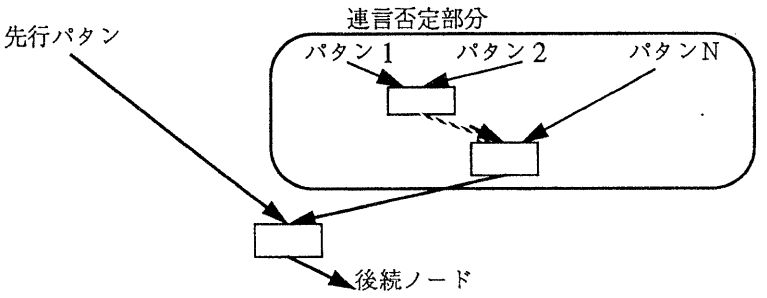

図 8. 連言否定を副木にした Rete ネット表現

1) 構造が複雑である
元来のReteネットでは、右側のノードは常にαメモリにつながる形態になっており、これは4.1.でも同様である。しかし、副木構造を使う場合は、右側にもβメモリが出現することになる。このことは、2入力ノードテストの拡張を必要とし、これに対応するためのインタプリタおよびパーザの改造は、4.1.の手法が要求するものよりも大であると思われる。

2) 変数の制約を表現しにくい
連言否定全体と先行パタンとのパタンマッチを行なうことになるが、この場合、先行パタンに記述されている変数の制約を連言否定内部に伝えにくい。図9のような場合に、連言否定中の手続き的条件が、先行パタンの変数を参照した場合には、これに対応できない。この点、4.1.では、連言否定の各パタンは逐次先行パタンに連結され、変数の制約を伝えることができる。

```
(p rule1
   (a ^a1 <x> ^a2 <y>)
   -( (b ^b1 <y>) : (> <x> 3))
--> 右辺)
```

図9. 副木で対応できないルール例

5. おわりに

変数表現と否定表現におけるReteアルゴリズムの制約を述べ、それらの制約を解消する方法を提案した。Rete自体は、PS処理系の実現手段として広く用いられてきたが、その文法的な特性を論じた報告はこれまで少なかったように思われる。現在でも、プロダクションシステムはエキスパートシステム構築のための主要な手段である。プロダクションシステム処理系に関する研究では、従来はその速度性能を改良することに焦点が集中していた。今後は、システム構築言語としての記述力を強化することも主要な課題であるといえる。

謝辞

Reteの理解において御指導いただいた福岡工業大学教授荒屋真二先生、本案の実現に御協力いただいた米国Horizon Research Inc.の諸氏、および本研究に貴重な助言を頂いたICOT太田好彦氏に深く感謝致します。

参考文献

1) Forgy,C.L.:Rete:A Fast Algorithm for the Many Pattern/ManyObject Pattern Match Problem*, Artificial Intellegence 19, pp.17-37,(1982).
2) 荒屋真二、他:プロダクションシステムにおける効率的パタン照合のための連想Reteネットワーク表現, 情処学会論文誌 Vol.29,No.8, pp.741-748,(1988).
3) 田野俊一、他:ST-NETアルゴリズム：双方向推論の高速処理方式, 情処学会論文誌 Vol.29, No.8,pp.741-748,(1988).
4) Gupta,A.:Results of Parallel Implementation of OPS5 on the Encore Multiprocessor,CMU-CS-87-146, Carnegie-Melon Univ.,(1987).
5) Miranker,D.P.:TREAT:A Better Match Algorithm for AI Production Systems,AAAI'87,pp42-47,(1987).
6) Miranker,D.P.,et al.:On the Performance of Lazy Matching in Production Systems,AAAI'90,pp.685-692, (1990).
7) Forgy,C.L.:OPS5 User's Manual,CMU-CS-81-135,Carnegie-Melon Univ.,(1981)