7D-2　汎用エキスパートシステムの高速化技術(Ⅰ)　開発構想

黒沢　憲一　　島田　優　　平山　洋一
日立製作所　日立研究所

1．はじめに

近年、プロダクションシステム技術を用いたエキスパートシステムの開発が盛んである。その中核である推論エンジンの特徴は、RETEアルゴリズムをさらに改良していること、及びC言語等の手続き型言語で記述することにより高速化を実現している点にある。しかしながら、これまでに開発されてきたエキスパートシステムの規模は数百～数千ルール程度であり、今後は知識がより複雑化し、システムの規模は数千～数万ルールと一桁以上巨大になることが予想される。これに伴い推論性能の高速化要求は、さらに高まるものと思われる。そこで本報告では、高速化を追及した新しいコンパイル型のプロダクションシステム高速処理方式を提案する。

2．従来の高速推論処理方式

2．1プロダクションシステムの概要

プロダクションシステムは、条件照合（パターンマッチ）、競合解消、実行の3つのサイクルを繰返しながら、実行可能なルールが存在しなくなるまで推論を行うことを基本にしている。このためRETEアルゴリズムでは、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グラフに変換し、そのノードに条件判定コマンドを配置し、グラフを最適化することにより複数ルール間に出現する同一の条件判定を削除したりまた、前のサイクルでの計算の中間結果を全てネットワークに保存し、ルールの実行に伴う変更分だけを対象に照合を限定することにより高速化を実現しているしかしながら、このように効率的なRETEアルゴリズムもその実現方法となると、ネットワークのノードをリストやテーブルなどの中間語形式で表現し、インタプリタが解釈実行する方法を用いていたため十分な性能を引き出すことができないという問題があった。このため最近ではRETEのネットワークを直接機械語命令に変換するコンパイル方式が採用されている。

2．2従来の高速推論方式の問題点

RETEアルゴリズムのごとく、更新されたフレームを含む中間結果がネットワーク上に多数存在する場合には、その中間結果を全て削除し、再照合時に新たに中間結果を追加しなければならない。すなわち、メモリ管理のオーバヘッドが大きいという欠点がある。例えば第1図のルールとフレームから成るプログラムをRETEネットワークに変換すると第2図のようになる。このプログラムのルール1のif部は、”クラス名がfbのインスタンスフレームに対し、その属性dの値を変数YXへ代入し、かつクラス名がfaであるインスタンスフレームに対しその属性bの値が10より小さく、かつそのフレーム名称を変数Zに代入するならば”という意味で、then部以下がルール1の実行部である。ルール2も同様なので説明は省略する。またフレームは、クラスフレームとしてfa,fbの2つがあり、そのインスタンスフレームとしてfa1,fb1,fb2が存在する。インスタンスフレームの属性の初期値はクラスフレームのものと同一である。このプログラムの特徴をRETEアルゴリズムの観点からみると、ルール1、2間の条件が類似しているため共通条件を共有化した効率の良いRETEネットワークへ変換可能な点、およびルール条件部で参照しているフレーム名が変数となっているため、ネットワーク上に保存される中間結果が複数個生成されるので、パターンマッチ回数を大幅に削減できる点にある。実際、第2図のRETEネットワークでは、ルール1の第1条件とルール2の第2条件、およびルール1の第2条件とルール2の第1条件が共有化されているため、条件判定結果が、中間結果としてネットワーク上のSUB_TERM，MERGEノードに保存される。この結果、ルール条件部で参照しているフレームが更新された場合の条件判定回数は、大幅に削減可能である。例えば、インスタンスフレームfa1が変更されるとルール1の第1条件のINTRA1で照合判定が1回行われるが、ルール2ではその中間結果をSUB_TERMから参照するだけなので、照合判定は行われない。このため合計1回の照合判定回数で済んでいる。

しかしながら、このプログラムはRETEにとって最も性能低下の激しいプログラムの1つとされている。それは、照合判定回数は削減できたが、中間結果の更新に多大の時間を要するためである。例えば、fa1が更新された場合、RETEネットワーク上のルール1における2つのSUB_TERM,MERGE,RULE_TERMに保持されたfa1を含む全ての中間結果を検索して削除し、かつ新たな中間結果をネットワーク上に作成する処理を行う必要があるからである。

このように中間結果をネットワーク上に保存することは、良い面も悪い面も持ち合わせており、単純に条件判定回数のみをアルゴリズムの評価指標とするのは危険である。すなわち、RETEアルゴリズムの評価には、条件判定回数と中間結果の更新量の2つを評価指標としなければならない。

またRETEアルゴリズムにおけるもう1つの問題は、その実現方式である。RETEは、データフローモデルに基づいてネットワークに更新フレームを”流し込む”というデータ駆動型の実行メカニズムを採用しているため、通常のパイプライン計算機上での実現には、デ

A Performance Improvement Technology in Expert Systems(Ⅰ) Design Philosophy,
Kenichi Kurosawa, Masaru Shimada, Hirokazu Hirayama,
Hitachi Research Laboratory, Hitachi Ltd.

ータフロー計算機を仮想的にインタプリタプログラム（推論エンジンプログラム）で実現し、ネットワーク上に配置された仮想命令をインタプリタによって解釈実行しなければならなかった。このため推論は、仮想命令をインタプリタを介して実行する必要があるため効率良く実行できないという問題があった。このため従来技術の高速化方式として、RETEネットワーク上の条件判定コマンドを機械語命令列に変換することにより高速化する方式が存在する。しかしながらフレーム数が数千～数万にも及ぶ大規模システムではフレーム自体がリストやデータなどテーブルの形式で表現されているため、フレームを参照するために何度もポインタをたどることが多く、フレーム参照ネックにより十分な推論性能を実現できないという問題があった。

3．コンパイル型推論処理の高速化方針

上記問題点を解決して高速推論を達成するため、以下の２項目を高速化方針とした。

（１）解釈オーバヘッドの削減

・ルールだけでなくフレームも共に機械語命令列に変換して実行する。

（２）パターンマッチ回数の削減

ルールの各条件ごとに、

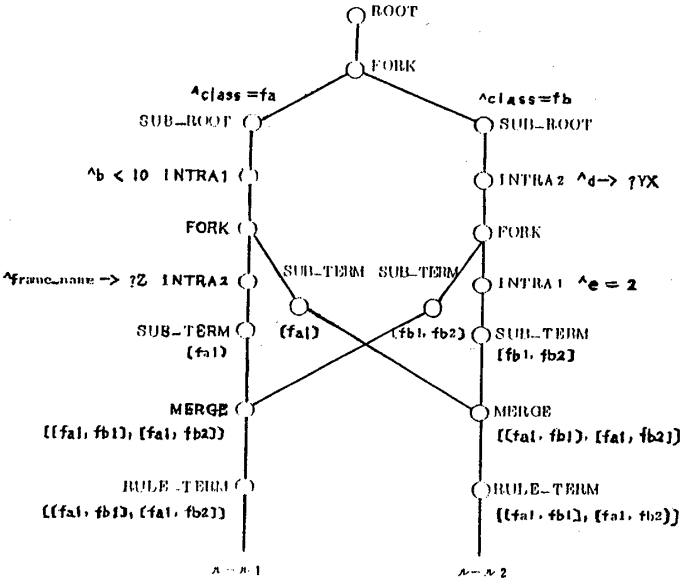
・フレーム名称が明示されている定数フレーム条件の場合は中間結果を保存する。

・フレーム名称が明示されていない変数フレーム条件の場合は中間結果を保持せずにハッシュ技法により参照すべきフレームを絞り込む。

すなわち、本報告で述べる方式はコンパイルすることを前提にしているため高速に照合できる点に着目した方式である。例えばルール条件部で記述される照合演算子は単純な比較、代入であることから照合判定に要する時間はそれほどかからない。このためルールとフレームを共に機械語命令列へ変換して高速化を図る方針とした。しかしながら、フレームも含めて機械語命令列に変換してコンパイル型で前向き推論を実現する方法は未だ存在せず、フレームの命令表現方法、ルール条件部の命令表現方法、さらにこれらの機械語をコントロールして全解探索を行う方法、フレームの生成・削除方法等を、いかに実現するかが大きな問題である。一方、ルールの条件が変数条件である場合は、実行すべきルール群と参照すべきフレーム群を出来るだけ絞り込まなければ照合判定回数は指数関数的に増加してしまうので、ハッシュ技法により実行すべきルールと参照すべきフレームを徹底的に絞り込む方針とした。また、定数フレーム条件が複数ルールの共通条件である場合は、固定した領域に中間結果を生成し、その中間結果を共有化することとした。このため従来のインタプリタ方式におけるRETEアルゴリズムとは異なり、中間結果を保存すべきケースと保存しないほうが良いケースが存在し、そのトレードオフを決定することが重要となる。以上から、ルールとフレームを共に機械語命令へ変換して推論するコンパイル方式は、RETEに比較して参照回数は増加するが、一方中間結果の削除、追加処理および中間結果を保持するためのメモリ管理処理は大幅に削減することができるという特徴をもつ。

参考文献

[1]石田、桑原：プロダクションシステムの高速化技術、情報処理、vol 29,No.5,pp.467-477,1988.



第２図　RETEネットワーク

```
(rule1 if
  (?X  ^class = fb
       ^d -> ?YX  )
  (?Y  ^class = fa
       ^b < 10
       ^frame_name -> ?Z)
       then .....          )
(rule2 if
  (?X ^class = fa
      ^b < 10       )
  (?Y ^class = fb
      ^e = 2
      ^d -> ?YX     )
      then .....          )
```

```
(class fa  ^super_class system
           ^b      1          )

(class fb  ^super_class system
           ^d      1
           ^e      2          )

(fa1       ^class      fa     )
(fb1       ^class      fb     )
(fb2       ^class      fb     )
```

第１図　ルールとフレーム