計算機アーキテクチャ 74－5
（1989. 1. 26）

ＩＰＰによる汎用エキスパートシステムの高速化技術

島田 優 ， 黒沢 憲一 ， 平山 洋一
（株）日立製作所 日立研究所

代表的な高速推論方式であるＲＥＴＥアルゴリズムは照合回数は少なくなるが、中間結果の管理のために中間結果の更新が多い場合必ずしも有効でない。そこで本報告では、逆に中間結果を使わずに推論を高速化する方式を提案する。この方式では中間結果利用の効果をフレーム参照の高速性で補い、中間結果管理の手間を無くす。そこでフレームをスロット単位で命令化し管理するコンパイル方式を提案し、フレーム参照の高速化を図った。そしてＩＰＰのハードウェアを活用してこの方式で有効な専用命令を定義した。簡単なプログラムを用いてこの専用命令を評価した結果有効性を確認し、ＩＰＰ上での性能を従来のインタプリタ方式と比較した結果本コンパイル方式により約３０倍性能を向上させる見通しを得た。これによりＲＥＴＥのように中間結果を用いずとも推論を高速化できることを明らかにした。

A FAST ALGORITHM FOR EXPERT SYSTEMS USING IPP

Masaru SHIMADA , Ken'ichi KUROSAWA and Hirokazu HIRAYAMA

Hitachi Research Laboratory, Hitachi Ltd.
4026, Kuji, Hitachi, Ibataki, 319 Japan

RETE algorithm which is most famous fast pattern matching algorithm, darstically decreases the number of matching between condition elements and working memories. But its mejor defect is inefficient processing of temporal redundancy resulting pattern matching. Therefor, we propose a new method for fast inference without temporal redundancy. The main idea is to compile working memories into special and general instructions of the Integrated Prolog Processor (IPP) for fast reference of values. Performance evaluation of IPP using simple programs show that these special instractions are effective for fast inference, and that the inference of the compiler is about 30 times as fast as that of an interpreter based on the RETE algorithm.

1.　はじめに

近年様々な分野においてエキスパートシステムが広く普及しつつある。これに伴い、エキスパートシステムに対する高速化の要求も高くなってきている。例えば診断型エキスパートシステム等の分野ではより大規模な知識を扱うようになってきている。しかし知識が大規模になると一般に推論性能が大幅に低下してしまう。そのため大規模な知識を扱うシステムではより高速な推論性能が必要とされる。またリアルタイム処理分野におけるエキスパートシステムも、より速い応答性が必要なシステムに適用するため、推論性能の向上が望まれている。このようにエキスパートシステムの高速化に対しては高い要求がある。

本稿ではこの要求に応えるため、高速推論方式を提案し、その性能評価を行なう。

まず、従来の代表的な高速化技術の問題点を分析する。そしてこの問題点を解決する高速化技術として、照合の中間結果を記憶せず、状態の変化に即座に対応可能な推論方式を提案する。次により高速な推論を行なう為に、この高速推論方式を用いた新しい知識のコンパイル技術について述べる。そしてこのコンパイル型推論方式に有効な専用命令を提案し、ＩＰＰ上でその推論性能を分析、評価する。

2.　従来の高速推論方式

本章では先ず、これまで大部分のエキスパートシステムで採用されてきたプロダクション・システムについて概略を述べる。次に従来提案されてきた代表的な高速推論アルゴリズムについて検討を行ない、その問題点を分析する。

2.1　プロダクション・システムの概要

プロダクション・システムは専門家の知識を　ＩＦ　～　ＴＨＥＮ　…　という形式で表現したプロダクション・ルール（以下ルールと呼ぶ）と推論対象の状態を表現する作業記憶とを用いて推論を行なうシステムである。通常これらのルールで　ＩＦ　と　ＴＨＥＮ　の間に記述された部分を　ＬＨＳ　(Left Hand Side)　と呼び、ＴＨＥＮ　以降に記述された部分を　ＲＨＳ　(Right Hand Side)　と呼ぶ。

プロダクション・システムは後向き推論を行なうシステムと、前向き推論を行なうシステムの二つに大きく分けられるが、本稿ではこのうち前向き推論について検討を行なう。前向き推論の場合、ルールのＬＨＳはそのルールの実行条件を表し、ＲＨＳはルールを実行する場合の動作を表す。推論は認知-行動サイクルを繰り返すことにより行なわれ、ＬＨＳと作業記憶との照合を行ないＬＨＳを満足する作業記憶とルールの組合せ（インスタンシエーション）を全て探し競合集合を生成する。次にこの競合集合の中から特定の戦略に基づき、インスタンシエーションを一つだけ選び、選ばれたルールのＲＨＳを実行して作業記憶を更新する。そして更新された作業記憶を使い新たに競合集合を生成し直す。一つの作業記憶に対するＬＨＳの照合条件を条件要素と呼び、一般にこの条件要素は各作業記憶の複数の属性値に関する条件項目から成り立っている。照合は各ルールの条件要素と作業記憶とのパターンマッチにより行なわれるため、推論時間は本質的にルール数と作業記憶数との積に比例してしまう。

そこで、従来よりこの照合動作の高速化に関して様々な研究が行なわれてきた。McDermottらは

1）条件要素間の関係 (Condition Membership)
2）どの条件要素をどの作業記憶が満足させているかの関係 (Memory Support)
3）ルール中の条件要素間の関係 (Condition Relationship)

に関する情報を組合せることにより、条件の照合を行なうルールを絞り込み、効率良く照合動作を行なうことを提案した。これらの関係を用いた著名な高速化手法として　ＲＥＴＥ　ｍａｔｃｈ　アルゴリズム[1]とＴＲＥＡＴアルゴリズム[2]がある。

2.2　ＲＥＴＥアルゴリズム

Ｒｅｔｅ　ｍａｔｃｈ　アルゴリズム（以下ＲＥＴＥアルゴリズムと呼ぶ）は現在多くのエキスパートシステム構築ツールが採用しており、

1）各認知-行動サイクルにおいて作業記憶の大部分は変化しない（一時冗長性）
2）各ルールの条件要素は互いに類似したものが多い

という仮設を前提とした、推論の高速化方式である。ＲＥＴＥアルゴリズムではルールのＬＨＳをＲＥＴＥネットワークと呼ばれる一種の弁別ネットワークに変換する。このネットワークのノードは条件要素の判定項目に対応しており、それぞれのノードには判定が成功した作業記憶に関する情報を中間結果として保存しておく。そして変化した作業記憶をトークンとしてこのネットワークに流し、ネットワークを更新することにより照合動作を行なう。このようにしてＲＥＴＥアルゴリズムでは一時冗長性を利用して、各認知-行動サイクルで毎回全てのルールと作業記憶との照合を行なわず、ネットワークの各ノードに保存されている前サイクルの中間結果を更新することにより条件判定の

回数を減らしている。RETEアルゴリズムではこの他に共通した条件要素を統合してネットワークを共通化し、条件判定の回数を減らしている。また田野俊一ら[4]は条件判定の順序変更やネットワークの変形などにより、このRETEアルゴリズムを改良した高速推論方式を発表している。

2．3　TREATアルゴリズム

RETEアルゴリズムが利用している一時冗長性はすべてのシステムに現れるわけではなく、リアルタイム処理等では作業記憶が一時に、大量に変更されてしまう場合も多い。これは中間結果の大巾な更新を引き起こし、推論性能低下の大きな原因となる。そのためTREATアルゴリズムではネットワークに保存する履歴として、一つの条件要素内で完結する判定項目の結果（α-memory）のみをノードに保存する。一方変数を介して条件要素間にまたがった判定項目の整合性に関する判定（join）の結果（β-memory）は保存しない。そのかわりに、各認知－行動サイクル毎にjoinの最適化を行ない、計算しなおす。これにより、一時冗長性の無いシステムにおいても中間結果の更新量が削減される。

2．4　従来方式の問題点

上記のごとく、従来の代表的な高速推論方式であるRETEアルゴリズムにおいては作業記憶の変更が多数発生した場合、中間結果の変更が頻発しこの中間結果の更新によるオーバーヘッドのため推論性能が向上しない。またβ-memoryは、更新に多くの手間がかかる上、これは部分的な照合結果であるために無駄になることが多い。一方TREATアルゴリズムのようにβ-memoryを持たない場合でも、中間結果のjoin操作の負荷は依然重く、更に動的にjoin操作の最適化を図るための操作量も多くなる。また中間結果を残す方式では動的メモリ管理が複雑になる。さらに作業記憶の更新が多く発生するようなシステムにおいて、ルールにおける条件要素中に変数が多く現れ条件要素間で互いに影響を与える場合、join操作が多くなり推論性能の低下が激しい。

3．　新推論方式

以上のようにルールの実行により変更になる作業記憶が少なければ中間結果などのMemory Supportを用いることにより推論を高速化することが可能である。しかし作業記憶の変更が多い場合にはかえってこの中間結果の更新がネックになり、推論性能が低下する。特にRETEアルゴリズムのように、この中間結果を活用しているものではこの欠点が大きく現れる。そこで本稿では従来有効であると考えられてきたこのMemory Supportを用いず競合集合のみを保存し、むしろ従来不利と考えられてきた条件要素と作業記憶との照合を直接行なう方式により、どれだけ推論の高速化が図れるかを検討する。中間記憶を用いずに競合集合のみを残す推論方式としては、既に新谷虎松［3］が提案した、プロダクション・システムのプログラムを部分計算技法により一度Prologプログラムに変換した後このPrologプログラムを実行して推論を行なう方式において用いられている。しかしこの方式では競合集合の管理や作業記憶の変更は、assert,retractなどの組み込み述語により行なうため、処理の負荷が大きくなってしまう。また一度Prologプログラムに変換するため、プロダクション・システムにとって余分な動作が発生する。

本方式のように中間結果を用いない場合の推論特性はRETEアルゴリズムとは逆に、作業記憶の変更が多いほど効率が良いと考えられる。一方作業記憶の更新が少ない場合、無駄な条件判定が多くなってしまい推論性能が低下する。そこで本方式においては個々の条件判定を高速化することにより推論性能の低下に対応する。また照合回数を出来るだけ減らすために予めルールの絞り込みを行う。

3．1　知識表現

まず方式を適用するシステムについて簡単に説明す

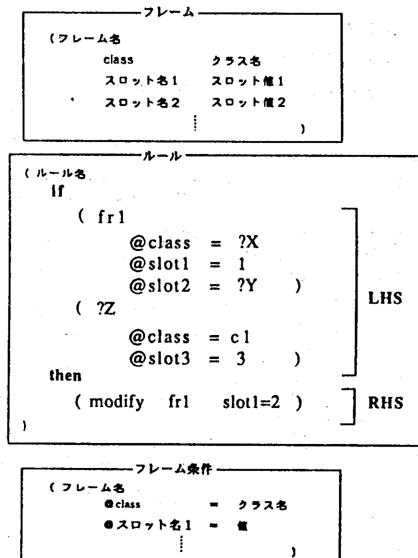


図-1　知識の表現

る。知識は図１のように作業記憶に対応するフレームと、ルールとで表現される。フレームは各々のフレームを区別するためのフレーム名、フレームの構造を決定するクラス名、属性に相当するスロットとから成り立っている。一方ルールに関しては、ＬＨＳの条件要素はフレーム条件に相当し、このフレーム条件は一般にフレーム名に関する条件、クラス名に関する条件、スロット値の条件から成り立っている。各条件には変数(？Ｘ,？Ｙなど）を用いることができ判定内容を動的に決めることが可能となっている。

３．２　競合集合の更新

中間結果の更新作業による推論性能の低下を避けるため、本方式ではＲＥＴＥアルゴリズムのように履歴として各条件項目ごとに中間結果を残す方式は採用しない。そして以前の照合結果として競合集合のみを保存しておくことにより、履歴の更新によるオーバーヘッドを最小限に抑えることにする。また中間結果を利用する方式では競合集合の更新は中間結果を組合せてインスタンシエーションを生成することにより行われる。しかし中間結果更新の手間が多い場合、条件要素と作業記憶との照合の手間が十分に小さければこの中間結果利用の効果は無くなってしまう。そこで本方式では、再照合時に高速に作業記憶を参照することにより照合の高速化を図る。そして順次インスタンシエーションを求めて競合集合に追加していく。この競合集合は各インスタンシエーションの集合であり、各インスタンシエーションはそのインスタンシエーションが属するルールのＬＨＳを満足させた作業記憶に関する情報を記憶させる。

３．３　高速全解探索

本方式では判定項目と作業記憶との高速な条件判定

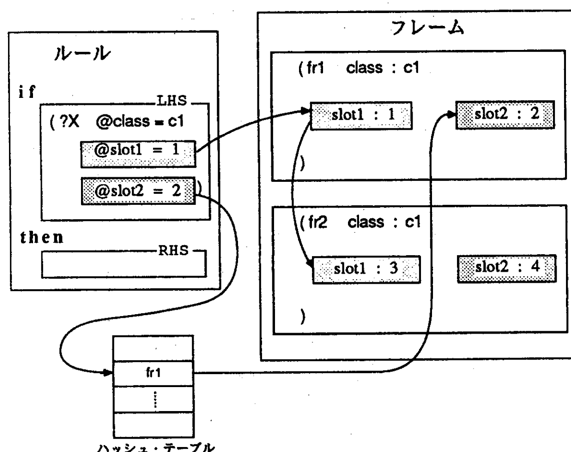


図-２　スロットの管理

を行うために、図２のようにフレームをスロット単位で管理し不必要なスロットの参照を避ける。そして条件要素でフレームを参照する場合には、その条件要素を構成する判定項目に対応したスロットを参照して必要な情報のみを参照する。条件要素でクラス名のみが指定されフレーム名が不定の場合には、参照するフレームが動的に決まる。そこで判定するスロットの値を高速に参照するために、各スロット毎にフレーム名によるハッシュテーブルを設ける。そしてフレーム名が決定した時点でこのテーブルを用い、判定するスロットの値を参照する。

全解探索を行なう場合、再照合を行うルール数を減らすため、照合条件で探索するパターンを分析して探索木を作り、予め探索を絞り込んでおく。例えば図３に示すようにクラス名が c1 でフレーム名が fr1 であるフレームの slot1 の値が３に変更されたとき、照合しなおすべきルールはルール１とルール３でありルール２は再照合の必要は無い。そこで予め slot1 に関する探索木としてルール１とルール３をつないでおく。

またルールの条件要素でスロット値を参照する場合、図４のように参照すべきフレームが複数ある場合のためにフレームに関しても探索木を構築する。そして２番目以降のスロットに関しては１番目に参照したスロット条件により動的に決定されるため、ハッシュテーブルにより直接参照する。

全解を探索するために以上のルールとフレームの探索木を辿りながら探索を行なう。まずルールの探索木を辿る前に、ルールの後戻り制御に関する情報よりなるルールコントロールブロック（ＲＣＢ）をスタックに生成する。一つのルールの照合が終わるとＲＣＢから制御情報を回復し、このＲＣＢの次の後戻りに関する情報を更新する。ルール探索木の最後に来たときには制御情報を回復した後、このＲＣＢをスタックから削除する。

フレームの探索を行なう場合も同様に、フレームの探索木を辿る前に、フレームの後戻り制御関する情報から成るフレームコントロールブロック（ＦＣＢ）をスタックに生成する。フレームの照合が１つ終わるとＦＣＢから後戻り情報を回復し、このＦＣＢの次の後戻りに関する情報を更新する。そしてフレームの探索木の最後に来たときに制御情報の回復を行なった後このＦＣＢをスタックから削除する。

４．　コンパイル方式

従来のエキスパートシステムの大部分はルールや作

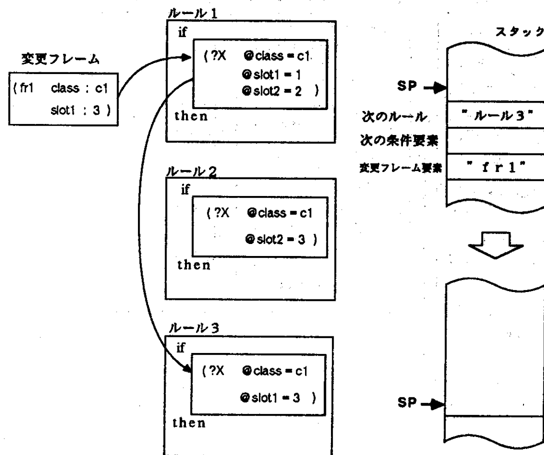


図-3　ルールの探索

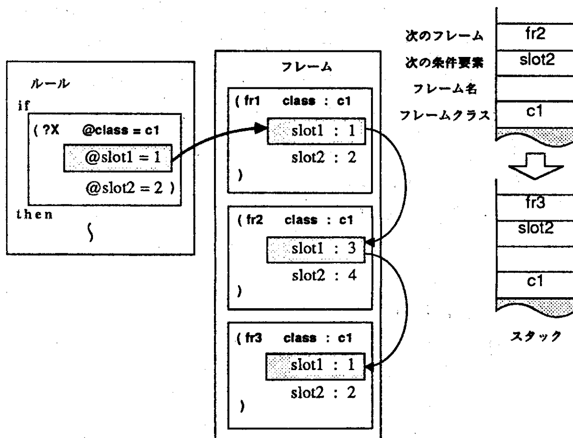


図-4　フレームの探索

業記憶はデータとして知識ベースに蓄え、実行時にはインタプリタがこれらのデータを読みだして解釈しながら推論を進めるためオーバーヘッドが大きかった。そこでより高速な推論を行なうため我々はコンパイル方式を採用することにした。

ＯＰＳ８３ [6] など従来のコンパイル型プロダクション・システムでは主にルールのみを命令に変換していた。しかし、本方式では中間結果を残さないためにフレームの参照が増えるので、推論性能向上のためにはフレームを高速に参照する必要がある。そのため本方式では従来データとしてのみ扱われてきたフレーム自体も命令に変換する。このときフレームはスロット単位で命令に変換し、これをスロットコードと呼ぶことにする。そしてある程度の判定はスロット自体に行なわせることにする。

次に本方式がどのように推論を進めるか図５を用いて説明する。まず競合解消により選択されたルールのＲＨＳを実行ＲＨＳテーブルを用いて検索し、ＲＨＳ命令を実行する。その結果フレームが変更された場合、ＬＨＳ選択テーブルを検索してＬＨＳ制御命令列を選択する。このＬＨＳ制御命令列は、ルールの探索制御命令の集まりであり、まずＲＣＢを生成し、ルールのＬＨＳを命令化したＬＨＳ命令列の１つを実行する。ＬＨＳ命令列ではフレームの変更により不要になったインスタンシエーションを削除した後、判定項目で照合するスロットの値を参照するためにフレーム制御命令列に分岐する。フレーム制御命令列では照合するフレームのスロットコードに分岐する。照合可能なフレームが複数ある場合、フレーム制御命令列ではスロット探索のためにＦＣＢを生成する。そして最初のスロットコードに分岐しスロット値を参照してＬＨＳ命令列に戻ってくる。同一フレーム条件中の２番目以降のスロット値は、参照するフレームの名前によりハッシュして値を参照する。２番目以降のフレーム条件の照合の場合も同様に、フレーム制御命令列に分岐してスロット値を参照する。この時タグを用いて参照したスロットの情報をチェックし照合を行う。照合時に判定が失敗した場合には、ＦＣＢから制御命令を回復し、ＦＣＢを更新した後フレームの別解を探す。全てのフレーム条件について照合が成功した場合、競合集合にインスタンシエーションを追加する。そして後戻り制御部分でＦＣＢから制御情報を回復しＦＣＢを更新した後、フレームの次候補との照合を行なう。フレームの次候補が無い場合にはＲＣＢから制御情報を回復してからＲＣＢの更新をした後、次のルールを探索する。ＬＨＳ制御命令列により指定された全てのルールを探索し終えると、再び競合解消部分に戻り競合解消を行なう。このサイクルが繰り返されて推論が進む。

５．　ＩＰＰと専用命令

コンパイル方式により知識を命令に変換するする際、ＩＰＰのハードウェアを活用した専用命令を定義し、これを用いることにより、より高速な推論の実行を可能にする。そこでまずＩＰＰの概要を述べた後、エキスパートシステム高速実行用の専用命令を提案する。

５．１　ＩＰＰ

ＩＰＰはスーパーミニコンに推論支援ハードウェアを内蔵したもので、８ＭＩＰＳの基本性能を達成している。

５．１．１　汎用ハード

ＩＰＰの基本部分である汎用ハードウェアは３２ビッ

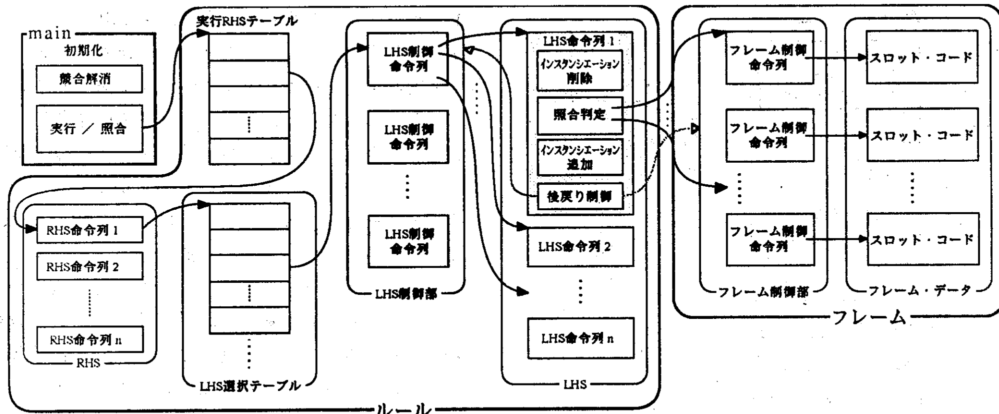


図-5　実行サイクル

トのアーキテクチャである。そして１６本の汎用レジスタを持ち、実行は多くの汎用機で採用されているパイプライン処理を採用している。このパイプラインは５段構成になっている。

５．１．２　専用ハード

ＩＰＰは図６に示すように従来アーキテクチャのハードウェアに加え大容量レジスタファイルを設けレジスタ本数を強化している。またタグの切り出し、タグの判定、タグの結合などのタグ処理ハードが付加されている。そして専用命令のためのマイクロプログラムを持っている。

今回のコンパイル方式において使用可能なレジスタの多いＩＰＰではより高速な実行が期待できる。また値の判定もタグ処理ハードを用いることにより高速化できる。

５．２　専用命令

ＩＰＰのハードウェアを活用してエキスパートシステムの高速実行に有効な専用命令を提案する。まず、本方式の場合ＬＨＳ制御命令、及びフレーム制御命令が多く発生するためこのコード量を減らすと共にキャッシュのヒット率を向上させるため、これらの制御命令を専用命令化する。また照合時のタグ処理やハッシュテーブルによる分岐も、パイプライン処理を途切れさせるため、これらの命令も専用命令化する。さらにＩＰＰでは使用可能なレジスタが増えているため、この増えたレジスタを操作するための命令を設ける。この命令を用いることによりメモリのアクセス回数を減らすことが可能となり、高速な推論を行うことが出来る。以下主な専用命令を提案する。

５．２．１　制御命令とスタック

本コンパイル方式ではルール、スロットの探索制御をそれぞれＬＨＳ制御命令列とフレーム制御命令列とで実現する。従って実際の推論時にはこれらの制御命令の実行が多いと考えられる。またこれらの制御命令列はＦＣＢ，ＲＣＢ操作のために多くの命令を必要とする上、ルールやフレームが増えるにつれて数が増加する。そこでこれら制御用の専用命令を定義しコード量の増加を抑える。

まずＬＨＳ制御のために

```
create_RCB  L
modify_RCB  L
delete_RCB  L
```

の命令を設ける。create_RCBはスタックにＲＣＢを生

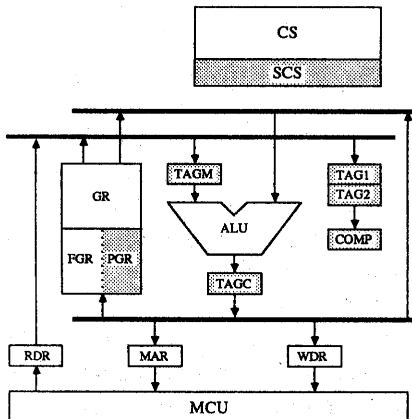


図-6　ＩＰＰハードウェア

成し、アドレスLに分岐する。modify_RCBはRCBから制御情報を回復し、RCBを更新した後アドレスLに分岐する。delete_RCBはRCBから制御情報を回復し、このRCBをスタックから削除した後、オペランドLで与えられたアドレスに分岐する。LHS制御命令列はこれらの専用命令が並んだものである。

フレーム制御のためには

```
create_FCB  L
modify_FCB  L
delete_FCB  L
```

の命令を設ける。create_FCBはまずFCBを生成し、オペランドとして与えられたアドレスLに分岐する。modify_FCBはFCBから制御情報を回復し、FCBを更新した後オペランドとして与えられたアドレスLに分岐する。delete_FCBはFCBから制御情報を回復し、FCBを削除した後オペランドとして与えられたアドレスLに分岐する。フレーム制御命令列はこれらの命令が並んだものである。

5．2．2　ハッシュ命令

本方式ではスロットの値を高速に参照するためにハッシュ技法を用いている。照合高速化のために

```
callc  Tsize,Reg,Ttop
```

というハッシュテーブルを用いてスロットの探索制御を選択する専用命令を定義する。これは
実行時にはRegで指定された値により、Ttopで指定されたテーブルを用いて分岐する。Tsizeはハッシュテーブルのサイズを表す。

またリアルタイム分野などでの実際のエキスパートシステムでは、フレームの値がかなり頻繁に変更される場合がある。そこでスロット参照命令の他にRHSでフレームのスロット値を高速に変更するために

```
hash_assign  Tsize,Reg,Ttop
```

というスロット変更命令を定義する。動作はRegで指定された値により、Ttopで指定されたハッシュテーブルを用いてスロット値を変更する。Tsizeはこのハッシュテーブルのサイズを表す。

5．2．3　型のチェックとタグ処理

参照したスロットの値が、比較する値と同じ型かどうかの型判定も含めた、タグによる判定項目の比較のために

```
lesth  Arg1, Arg2
grath  Arg1, Arg2
equal  Arg1, Arg2
```

等の比較命令を設ける。
またスロット情報のチェックのためにパターンマッチ命令として

```
ucsti  Arg1, Arg2
```

を定義する。

6　評価と検討

ここでは専用命令を用いてIPP上でテストプログラムを実行し評価した結果を示す。このデータのうちコンパイル方式に関するものは全てキャッシュのヒット率100％で計算した結果である。

6．1　ルール数とフレーム数

表1に2ルール／3フレームのプログラム（BENCH2）と2ルール／6フレーム（BENCH3）のプログラムを評価した結果を示す。これは同一のルールを用いてフレームの数を変化させた場合の、専用命令の出現率を調べたものである。評価に用いたプログラムはフレーム条件中のフレーム名が変数になっていて、RETEアルゴリズムの場合中間結果の更新が多いケースになっている。再照合にかかる時間を見てみると共に照合時間の約40％以上を専用命令が占めている。同様のことはBENCH2と6ルール／3フレーム（BENCH4)のプログラムの場合にも言える。BENCH4の結果を見てみると、この場合は照合時間の約50％を専用命令が占めている。このことは定義した専用命令の有効性を裏付けている。

6．2　専用命令と汎用命令

ルールが増えた場合でもフレームが増えた場合でも照合時間の約40％以上を専用命令が占めていることから、BENCH1について汎用命令と専用命令を用いた場合での性能差を見てみる。BENCH1は1ルール／1フレームのプログラムであり、ルールのフレーム条件が1つのものである。結果は図7に示すように汎用命令のみの場合を1とした時、専用命令を用いた場合BENCH1では約2倍の性能差になっている。BENCH1のプログラムでフレーム数を増やし、またフレーム条件の数を増やした場合の性能差を見てみると、汎用命令のみの場合を1とすると専用命令を用いた場合約3倍性能が向上している。このようにシステムの規模が大きくなり照合時間が増えると、この性能差はより大きくなると考えられる。

6．3　全体評価

最後に専用命令を用いた場合の本方式と、従来のインタプリタ型の代表的なプロダクション・システムと性能を比較してみる。結果は図8に示すように専用命

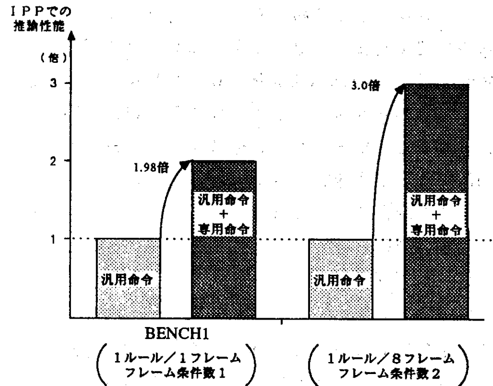
令を用いコンパイルした本方式は約２８倍の性能を達成している。このように従来高速化に有効であると考えられていた中間結果を用いなくとも、高速なプロダクション・システムの実現が可能であることが分かった。しかし、フレームの変更が少なくＲＥＴＥアルゴリズム等のように、中間結果が有効に利用できる場合などでは、かえって本方式では無駄な照合が多くなってしまう。従ってコンパイル時にルールの細かな分析を行ない、ＬＨＳ制御命令列で照合するルールを十分に絞り込むことが必要になる。また競合解消削除のための時間は全実行時間の３０％近くを占めていて、ルールやフレームが増えるとこの時間も無視できなくなる。そのため本方式でも競合集合操作の高速化を検討する必要がある。

7　おわりに

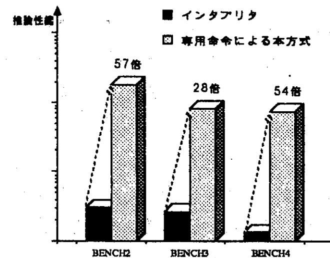
本稿では、一般的に推論の高速化に有効であると言われてきた照合の中間結果を用いず、逆に従来高速化には不利であると考えられていた、条件要素と作業記憶との直接照合をベースとする、プロダクション・システムの高速推論方式を提案し、その有効性を評価した。

まず中間結果を用いずに競合集合を更新するため、後戻り制御を用いた全解探索方式を提案した。次に中間結果を用いないことによる推論性能の低下を、作業記憶参照の高速化により補うため、従来データとして扱っていたフレームを命令に変換するコンパイル方式を提案した。さらにこの方式において有効な専用命令を提案し、評価を行ない有効性を確認した。そして本方式により中間結果を用いなくとも、高速なプロダクション・システムの実現が可能であることを明らかにした。

本稿で提案した方式は中間結果を用いるＲＥＴＥアルゴリズムの、いわば対極にあたるものである。今後は本文に記した欠点を補いさらに高速な方式を検討していく必要がある。



図－７　専用命令の効果



図－８　従来方式との比較

参考文献

[1] Forgy,C.L. : "RETE: A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem",Artificial Intelligence,vol.19 pp17-37,1982

[2] Miranker,D.P. : "TREAT: A Better Match Alogrithm for AI Production System,AAAI'87 pp42-47,1987

[3] 新谷虎松 : 推論エンジン KORE/IE,Logic Programming Conference '87,pp.233-242 1987

[4] 田野俊一,増井庄一,坂口聖治,船橋誠壽 : 知識ベースシステム構築用ツールEUREKAにおける高速処理方式，情報処理学会論文誌，Vol.28,No.12,pp.1255-1268,1987

[5] 石田 亨,桑原和宏 : プロダクションシステムの高速化技術，情報処理,Vol29,No.5,pp.467-477,1988

[6] Forgy,C.L. : OPS83 User's Manual and Report, Production Systems Technologies Inc. ,1985

表－１　推論時間の分析

	BENCH1		BENCH2		BENCH3		BENCH4	
	1ルール ／1フレーム		2ルール ／3フレーム		2ルール ／6フレーム		6ルール ／3フレーム	
照合サイクル	全実行時間比	専用命令実行時間比	全実行時間比	専用命令実行時間比	全実行時間比	専用命令実行時間比	全実行時間比	専用命令実行時間比
競合解消	13.31%	0.00%	6.37%	0.00%	2.79%	0.00%	1.61%	0.00%
RHS	25.65%	56.15%	9.88%	53.23%	4.32%	53.23%	4.12%	53.23%
競合集合削除	21.95%	0.00%	18.96%	0.00%	28.21%	0.00%	24.27%	0.00%
LHS	39.09%	40.00%	64.78%	46.74%	64.68%	40.25%	70.00%	52.08%
合計	100.00%	30.04%	100.00%	35.54%	100.00%	28.34%	100.00%	38.64%