コンパイル型プロダクションシステムの高速処理方式†

黒沢憲一†† 島田優††

本論文では，フレームをベースとするプロダクションシステムの高速化技術と実際のエキスパートシステムの分析結果について述べる．提案するコンパイル型推論処理方式は，従来のインタプリタ型やRETEネットワークをインライン展開した従来のコンパイル型とは異なり，ルールとフレームを共に命令列に変換して前向き推論を実現する方式である．その特徴は，コンパイラがレジスタを効率良く使用できること，フレームのスロットを高速に参照できること，複数のルールやフレームを効率良く実行制御できることであり，しかも，命令表現されたフレームを，推論実行中に生成，削除できる点にある．本コンパイル方式の効果を簡単なフレームとルールを用いてRETEアルゴリズムに基づくインタプリタと比較評価した結果，フレーム参照時間は8.5～14.5倍高速であり，ルール条件部におけるスロット値の演算は25倍，ルール実行部におけるスロット値更新は22倍高速であった．また，認知行動サイクル全体では，14～20倍高速であった．

1. はじめに

近年，フレームをベースとする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，および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フレームとルールの規模は数百～数千個程度であり，今後は知識がより複雑化し，その規模は数千～数万個と1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．そこで本論文では，特に，フレーム参照の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つのサイクルを繰り返しながら，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[1]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すべ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．

2.2 従来の高速推論方式の問題点

RETEアルゴリズムの長所は，更新されたフレームが前のサイクルでの中間結果にさほど含まれていないケースにおいて，高速かつ効率的にパターンマッチを実現できる点にある．すなわち，条件判定回数を大幅に削減することができる．しかしながら，逆に多く含まれている場合には，フレームが更新されると，その中間結果を削除し，再照合時に新たな中間結果をRETEネットワークへ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非常に大きいという欠点がある．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，推論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つを評価指標とすることが望ましい．

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，ルール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，解釈のオーバヘッドが原因となり，効率良く実行できないという問題があった．それゆえ，従来技

† High Speed Processing Method for a Production System Compiler by KENICHI KUROSAWA and MASARU SHIMADA (The 8th Department, Hitachi Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所日立研究所第8部

術の高速化方式には，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[2)~5)]．しかしながら，ルールのみを命令列に変換しただけでは，フレーム数が増加すると，ルール条件部からのフレーム参照は，そのフレームを一意に決定できるフレーム番号を基に，その番号を索引としたテーブルを介してそのフレームのスロット値等を有するフレームデータテーブルのポインタを求め，次に，参照したいスロットの文字列と，フレームテーブル中の複数のスロットの文字列を次々に比較し，一致したスロットの値をルールへ渡すという手順で行っていた．しかも，フレームが生成・削除された場合に対応できるように，フレーム間をポインタで結ぶという柔軟なデータ構造が必要であった．このため，フレーム参照は，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．

以上述べたごとく，フレームをベースとするプロダクションシステムの高速化には，

（1） パターンマッチ回数と中間結果の更新量の2つを評価基準とした推論アルゴリズム[6)~8)]

（2） フレーム参照の高速化を実現するコンパイル型推論処理方式

の2項目を実現することが重要である．ここで(1)の推論アルゴリズムについては，文献6)～8)にてその一実現方法が述べられている．

そこで本論文では，特定の推論アルゴリズムに限定せずに(2)のコンパイル型推論処理方式について述べる．

3. コンパイル型高速推論処理方式

3.1 高速化方針

我々は，フレーム参照の高速化を達成するため，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コンパイル型推論方式を提案する．

すなわち，本論文で述べる方式は，ルールとフレームを共に命令列にコンパイルすることにより，フレームの参照時間を短縮することが最大の特徴である．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前向き推論を実現する方法は存在せず，また，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．

3.2 エキスパートシステムの分析

まず，実際のエキスパートシステムではどのようなルールを用いているのか，約100ルール程度のエキスパートシステムについて調査を行った．ここで，エキスパートシステムAは，列車事故時のダイヤ再編成を行う計画型エキスパートシステムであり，Bは鉄板から部品の切り出し手順を指示するガイダンスエキスパートシステム，Cは電力系統の異常箇所とその原因を実時間で判断するリアルタイムエキスパートシステムである．このように，異なる分野のエキスパートシステムを用いて，その特徴を調査した．その結果を**表1**に示す．これによると，エキスパートシステムAのルールには，フレーム名が直接指定された条件節が122個あり，一方フレーム名が具体的に指定されず，代わりにクラス名とスロット名が指定されている条件節（すなわち，そのクラスに属するすべてのフレームを対象とした条件節）が79個もあった．本論文では，クラス/インスタンスフレームにかかわらず，唯

表1 エキスパートシステムのルール条件部分析結果
Table 1 Analysis of rule condition parts of expert systems.

エキスパートシステム	ルール数	ルール条件節数	ルールの条件部分析								
			条件節の種類と出現頻度				変数の出現頻度		変数の演算相手		
			フレーム名指定	クラス名指定	スロット名指定	合計	ローカル変数	共有変数	スロット値	フレーム名	クラス名
A	92	201	有	無	無	122	81%	19%	98%	2%	0%
			変数	有	有	79					
B	96	151	有	無	有	13	68%	32%	96%	3%	1%
			有	無	無	7					
			変数	有	有	131					
C	106	392	有	無	有	210	42%	58%	99%	1%	0%
			変数	有	有	144					
			変数	無	有	38					

一つのフレームを適用対象にしたルール条件節を定数フレーム条件節と呼び，また，その適用対象が“あるクラスに含まれるすべてのフレーム”のごとく，複数のフレームを対象にしたルール条件節を変数フレーム条件節と呼ぶことにする．同様に，エキスパートシステムBにおいても，定数フレーム条件節は 20 個であるのに対し，変数フレーム条件節は 131 個と圧倒的に多く，エキスパートシステムCでも変数フレーム条件節が約半数を占めていることがわかった．一方，条件節に出現する変数については，エキスパートシステムAでは他条件節に出現しない変数（ローカル変数と呼ぶ）は，すべての変数の 81% を占めており，他条件節にも出現する変数（共有変数と呼ぶ）は 19% である．しかしながらエキスパートシステムB，Cでは，ローカル変数がそれぞれ 68%，42% であるのに対し，共有変数は，32%，58% であり，共有変数が非常に多いという特徴がある．

以上の結果，エキスパートシステムの分野にかかわらず，ルール条件部は，変数フレーム条件節が多いこと，およびルール条件節中に出現する変数には，共有変数が多いという特徴を持つことがわかった．このことは，RETE アルゴリズムを用いた場合，多数の中間結果を生成することになるため，ルール実行部にて更新されたフレームが多くの中間結果に含まれる確率が高いことを意味し，また，多くの照合動作において，他の条件節との間で変数値の整合性を調べる必要があることも意味している．また，変数の照合相手は，そのほとんどがスロット値であるため，変数の値が更新されるとフレームのスロット値を参照し，照合を行う必要があり，特に，本質的に数多くのフレームを参照する必要のある変数フレーム条件節では，フレーム参照を高速化することが非常に重要であると言える．このように，プロダクションシステムのパターンマッチ処理を高速化するには，フレーム参照の高速化が重要であることがわかった．

一方，ルールの実行部に出現する文の種類について調べた結果を**表 2** に示す．エキスパートシステムAでは，既存フレームのフロット値を更新する assign 文が 79% を占めており，一方，入出力文と他言語呼び出し文が 21% を占めている．同様にエキスパートシステムB，Cにおいても assign 文は，それぞれ 54%，82% を占めており，推論実行中にフレームを生成する create 文や，削除する delete 文は出現していなかった．

表 2 エキスパートシステムのルール実行部分析結果
Table 2 Analysis of rule execution parts of expert systems.

エキスパートシステム	ルールの実行部分析					
	入出力文	assign 文	create 文	delete 文	他言語呼び出し文	合 計
A	5%	79%	0%	0%	16%	100%
B	20%	54%	0%	0%	26%	100%
C	6%	82%	0%	0%	12%	100%

以上の結果は，3種類のエキスパートシステムの分析結果ではあるが，より適応性に優れたシステムを構築するためには，変数フレーム条件節や共通変数が多いルールを記述することは自然であり，また，実行中にフレームを生成，削除することにより推論を行うよりも，既存のフレームのスロット値を更新して推論を行うほうが考えやすいため，assign 文が多くなるのは自然であると思われる．

そこで我々は，フレームの参照を高速化するため，フレームを命令列に変換することを基本とし，しかもフレームの生成，削除を可能とするコンパイル方式が最も望ましいと結論を得た．

3.3 フレーム高速参照を特徴とするコンパイル方式

高速なコンパイル型推論を実現するには，ルール，フレームの命令表現とそのインタフェース規約が重要である．そこで我々は，ルールからフレームを参照する時には，レジスタを介して必要なスロット値を参照するレジスタ渡しのインタフェースを基本とした．すなわち，ルール側では参照するフレーム名やクラス名を特定のレジスタへ代入してフレーム命令コードへ分岐する仕様とし，フレーム側はそのスロット値をスロットレジスタと呼ぶ特定のレジスタへ代入した後に再びルールへ戻る仕様とした．ただし実際には，レジスタ本数は有限個であるため，フレーム名やクラス名等の使用頻度の高いデータはレジスタへ割り当て，一方スロットは，メモリ上に作成した仮想レジスタへ割り当てることにより，十分な個数のスロットレジスタを確保することとした．これにより，ルールは高速にフレームを参照できる．

一方，前節でも述べたように，変数フレーム条件節では，中間結果を持っている場合でも共有変数が更新されると複数のフレームを参照しなければならないため，フレームの探索制御が必要であり，ルールについてもフレームが更新されると，そのフレームを参照し

ているすべてのルールを実行しなければならない．そこでまずフレームコントロールブロック（FCB）を用いたフレームの探索制御について述べる．**図1**と**図2**は，FCBの構成とその動作を示している．FCBのR1は，ルールで参照するクラスフレーム番号（第1レジスタの内容），R2はルールで参照するフレーム番号（第2レジスタの内容），NGRはルール条件部からフレーム命令列へ分岐する際に再びルール条件部へ戻るためのアドレス（ネクストゴールレジスタの内容），AFはフレーム命令列のアドレス情報を示している．この4種の情報から成るFCBを用いたフレーム探索制御の動作を図2を用いて説明する．すなわち，フレームの参照は，ルール条件部命令列からフレーム命令列（L1）へ直接分岐するのではなく，戻りアドレス（G）をNGRへ設定した後create_FCB命令列（C1）へ分岐してFCBをスタック上にストアし，その後にフレーム命令列（L1）へ分岐する．フレーム命令列（L1）ではスロット値を対応するレジスタへ代入した後にNGRで指示されたアドレスへ分岐する．この結果，ルール条件部命令列では，レジスタを参照することにより照合判定を行うことができる．次に他のフレームを参照する時にはレジスタSPで指示されたアドレスの内容（C2）を読み出し，modify_FCB命令列（C2）へ分岐することで実現できる．すなわち，この命令列ではFCBからレジスタの回復を行うと共にAFをC3に変更し，L2へ分岐する．これにより，次々とフレームを参照することができる．また，最後のフレームを参照する時にはdelete_FCB命令列にてFCBからレジスタの回復を行った後にFCBを消去することによりフレーム探索を実現する．次に，ルールの探索順序制御を実現するために用いるルールコントロールブロック（RCB）を**図3**に示す．ここでR4は，ルール実行部で更新されたフレームのフレーム命令列アドレス（第4レジスタの内容），NGRはルール実行部へ再び戻るためのアドレス，ARは次に照合を行うルール条件部の命令コードアドレスの3種の情報から成る．FCBと同様，スタック上にRCBを生成，更新，消去することによりルールの探索制御を実現できる．**図4**はRCBを用いたルール条件部命令列の探索制御動作を示している．その動作は基本的にFCBと同様なので詳しい説明は省略するが，各ルール条件部命令列ではレジスタR4で指示されたフレーム命令コードを実行することにより更新されたフレームの値を参照する点が異なる．このようにフレームとルールの探索はFCB，RCBを使って数回のメモリ参照と分岐命令で実現できるため，高速である．

次にルール条件部の命令コードを**図5**を用いて説明する．ルール条件部には，更新されたフレームを含むインスタンシエーションを中間結果と競合集合から削除する命令列と，フレームを参照して照合判定を行う命令列と，新しいインスタンシエーションを中間結果と競合集合へ追加する命令列と，他のフレームまたは他のルール条件部命令列を実行制御する命令列の4種から成る．このように，ルール条件部の命令コードは，中間結果と競合集合の

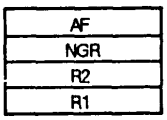

図1 FCBの構成
Fig. 1 Structure of FCB.

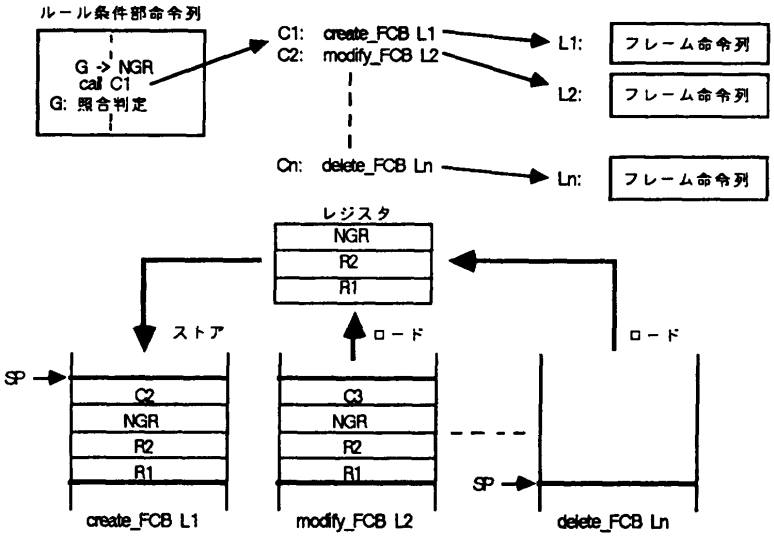

図2 FCBによるフレーム参照
Fig. 2 Frame reference using FCB.

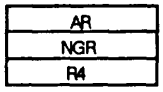

図3 RCBの構成
Fig. 3 Structure of RCB.

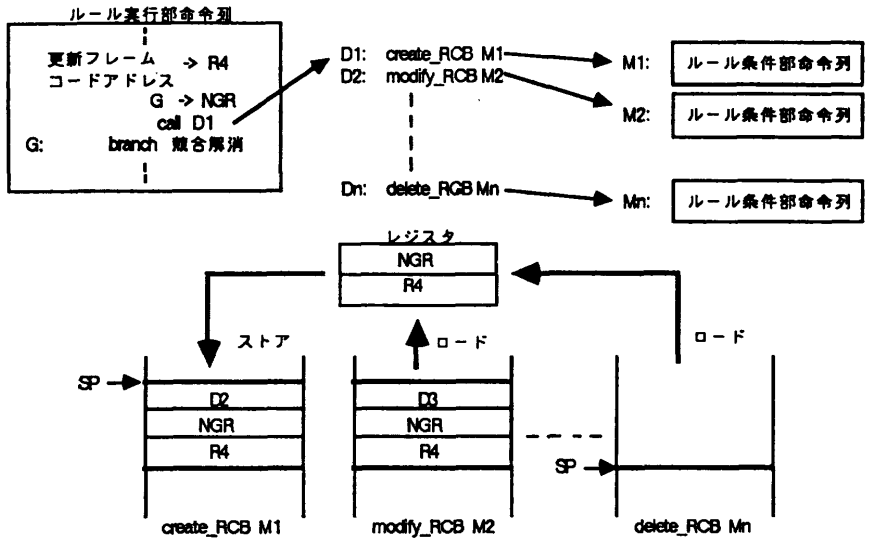

図 4 RCB によるルール条件部実行
Fig. 4 Execution of rule condition part instructions using RCB.

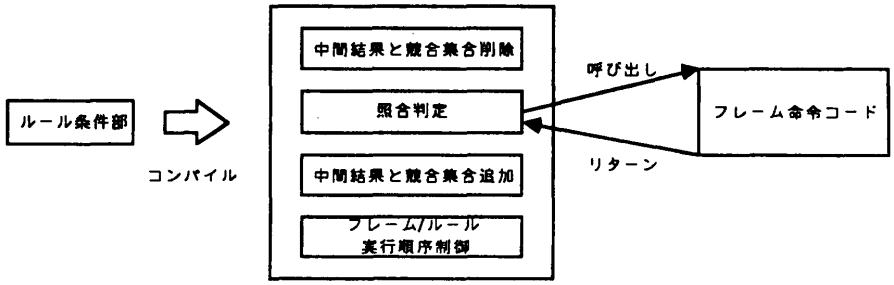

図 5 ルール条件部の命令コード
Fig. 5 Instruction codes for a rule condition part.

更新処理ばかりでなく，FCB と RCB を用いて次に参照すべきフレームまたは実行すべきルール条件部命令列を自動的に探索制御できる．

以上，フレームとルール条件部の探索制御方式について述べたが，探索をさらに高速化するには，その探索範囲をできるだけ小さくする必要がある．この点については，本節の最後に詳しく述べる．

次に，本論文で提案するフレームの命令コードについて述べる．フレーム参照を高速化するには，ルール条件節の種類に応じたフレームの命令コードを用意する必要がある．すなわち，ルールの定数フレーム条件節ではフレームが指定されているため，フレームの命令コードにそのクラスフレーム名やフレーム名などの情報がなくてもインスタンシエーションを作成可能である．一方変数フレーム条件節では，フレーム名が指定されないため，参照フレーム名やそのクラスフレーム名などの情報を当該フレームからルールへ渡す必要がある．すなわち，これらの情報がフレーム命令コードに必要となる．このためフレーム命令コードとして，定数フレーム条件節から参照する定数フレームコードと変数フレーム条件節から参照するための変数フレームコードの2種を用意して高速化を図ることにした．

まず，**図 6** の定数フレーム条件節と定数フレームコードの例を用いて，その動作を説明する．図からわかるように fa の定数フレームコードは，フレームの時間的な新しさを示すタイムタグ (time_tag) の格納アドレス time_adr を第3レジスタ R3 へ転送する命令と，スロット名称と一意に対応づけたスロットレジスタ SR へスロット値の格納アドレスを転送する命令列 (move slot1_adr, SR 1; move slot2_adr, SR 2; move slot3_adr, SR 3) と，最後にレジスタ NGR

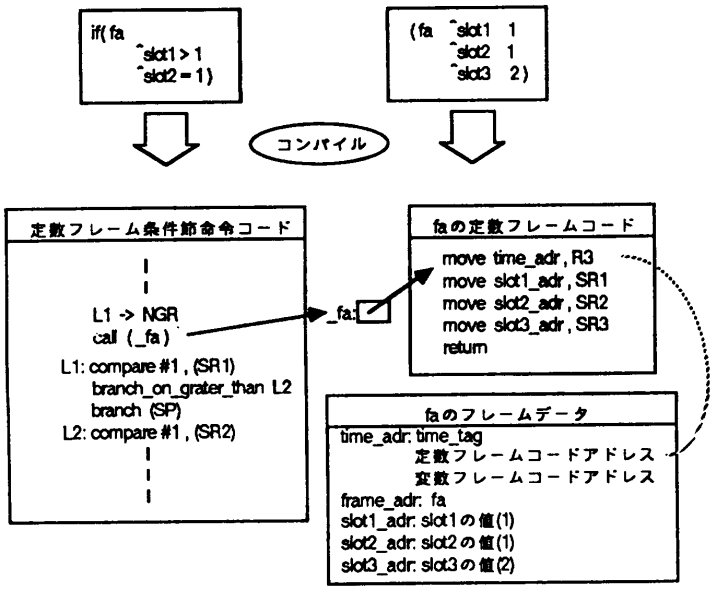

図 6 定数フレーム条件節と定数フレームコード
Fig. 6 Rule with a constant frame condition part and instructions for constant frame.

で指定された呼び出し元のルール条件節へ戻るリターン命令 return から成る．これにより定数フレーム条件節では，レジスタ NGR へ戻りアドレス（L1）を設定した後，_fa を参照して fa の定数フレームコードアドレスへ call 命令にて分岐し，定数フレームコードを実行することによりすべてのスロット値アドレス slot (*i*)_adr をスロットレジスタ SR (*i*) (*i*=1, 2, 3) へ代入した後に再び return 命令によって定数フレーム条件節の命令コードのアドレス L1 へ戻ることができる．次に compare 命令によって定数 #1 とスロットレジスタ SR 1 で指示された内容(1)と比較を行い，もし条件が成立すれば L2 へ分岐し，“slot 2=1” の演算を実行する．すべての条件が成立した時にはレジスタを参照してフレーム名とタイムタグアドレスとルール実行部で参照されている変数の値から成るインスタンシエーションを中間結果と競合集合へ追加することができる．逆に条件が成立しない時には，スタックの先頭アドレスを保持するレジスタ SP で指示されたアドレスの内容を読み出して分岐する．ここでもし，スタック上に RCB が存在すれば，次のルールを自動的に探索することが可能である．以上の結果，この例ではフレーム参照および照合判定が，10 命令足らず実現できるため非常に高速である．

ただし，本方式には定数フレームのスロット数が非常に多い場合に，性能が低下するという問題がある．しかし，前節のエキスパートシステムでは，フレームの平均スロット数は5個程度であり，しかも最近の計算機では，スロット値のアドレスをスロットレジスタへ代入する move 命令を1マシンサイクルで実行可能であるため，多少スロット数が多くてもそれほど問題にならないと思われる．

このコンパイル型推論処理方式のメモリ空間のイメージを図 7 に示す．このように本コンパイル方式は単一のスタックで推論を実現しているため，スタック管理以外の目的にレジスタを使用することができるという長所がある．また，インタプリタ型推論方式に比較し，フレーム命令コードが追加されたためオブジェクトサイズが大きくなるという欠点があるが，オンデマンドで必要な命令だけをロードして実行できる最近の計算機では特に問題ないと思われる．

次に，図 8 の変数フレーム条件節と変数フレームコードの例を用いてその構成と動作を説明する．変数フレーム条件節は複数のフレームを参照するため，変数フレームコードは，コンパクトな命令列であることが望ましい．そこで変数フレームコードの構成は，すべてのスロットを有した定数フレームコードとは異なり，1つのスロットだけを持つ構成とした．すなわち，*n* 個のスロットを有するフレームには，各スロットごとに *n* 個の変数フレームコードが存在する．ここでは図に示す fa の slot 1 の変数フレームコードを用いてその構成を説明する．fa の slot 1 の変数フレームコードは，そのクラス名 #f とフレーム名 #fa をそれぞれ R1, R2 と比較し（−1 は未代入変数フラグの意味），レジスタが −1 の時には #f や #fa を R1, R2 へ転送し，しかも fa のタイムタグアドレス (time_adr)，slot 1 のアドレス (slot 1_adr) をそれぞれ R3, SR 1 へ転送する命令列から成る．このよ

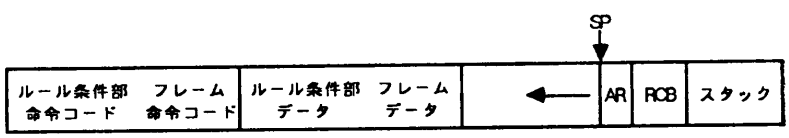

図 7 コンパイル型推論処理方式のメモリ空間
Fig. 7 Memory space for compiled-type production system.

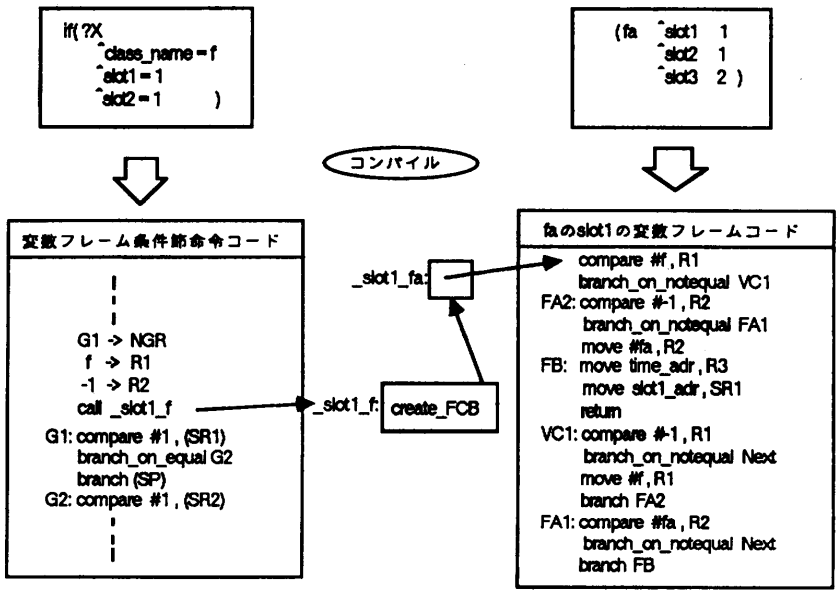

図 8 変数フレーム条件節と変数フレームコード
Fig. 8 Rule with a variable frame condition part and instructions for variable frame.

うに変数フレームコードは，クラスフレーム名とR1，および当該フレームとR2の比較命令列を含んでいる．これは変数フレームコード自身がルール条件部から参照されるべきフレームであるか否かを判断する機能を有していることを意味しており，毎回ルール条件部へ戻ってその判断をする必要がないという特徴をもつ．以下，動作を説明する．まず変数フレーム条件節では，戻りアドレス（G1）と参照すべきクラス名fおよびフレーム名が変数であることを示すフラグ（−1）をそれぞれ NGR，R1，R2 へ設定した後，call 命令にてアドレス_slot 1_f の create_FCB 命令列を実行し，その後に fa の slot 1 の変数フレームコードへ分岐し，再び return 命令によって G1 へ戻ることができる．この結果，各レジスタを参照することにより照合判定を行うことができる．また参照すべきフレームが複数存在する場合には，スタック上の FCB を参照してレジスタ R1，R2 の値を回復すると共に AF で指示されたアドレスへ分岐することにより次の変数フレームコードを実行できる．以上のことから，変数フレームの参照および照合判定は，10〜20 命令程度で実現できるため効率的である．

以上述べたルール，フレームを命令列に変換して前向き推論を実現する全体構成を図 9 に示す．ここで競合解消の命令列は，各ルールの競合集合を参照して競合解消戦略に従い実行すべきルールとそのインスタンシエーションを選択する．ここでインスタンシエーションの構成は，ルール番号，そのルールを発火可能にするフレームの組合せ，そのルール実行部に出現した変数の値の5種の情報から成る．次に，実行すべきルールの番号に基づきルール実行部テーブルよりルール実行部命令のアドレスを求めると共に，選択したインスタンシエーションのアドレスを第5レジスタ R5 へ転送してルール実行部命令列へ分岐する．この時戻りアドレスを NGR へ代入しておく．このようにして実行されるルール実行部命令列は，実行部で指定されたフレーム名を基にフレームデータテーブル（このテーブル中には，各スロット値とそのデータ型，そのフレームの時間的新しさを示すタイムタグ値，定数フレームコードアドレス，各スロットの変数フレームコードアドレス，フレームに付加された手続きのアドレス，上位クラス名，上位クラスから継承されたスロット値と手続きのアドレス等，当該フレームに関するあらゆる情報が格納されている）を見つけ，R5 を介してルール実行部に出現している変数の値を参照してスロット値を更新したり，フレームに付加された手続きを実行したりする．さらに，フレームのスロット値更新時に，フレームのタイムタグ値も更新し，その定数フレームコードアドレスを第4レジスタ R4 へ転送した後に，更新されたフレーム番号に基づきルール条件部テーブルを参照して create_RCB 命令列へ分岐することも実行命令列で行う．次に create_RCB 命令にて RCB 生成後にルール条件部命令列を実行するが，

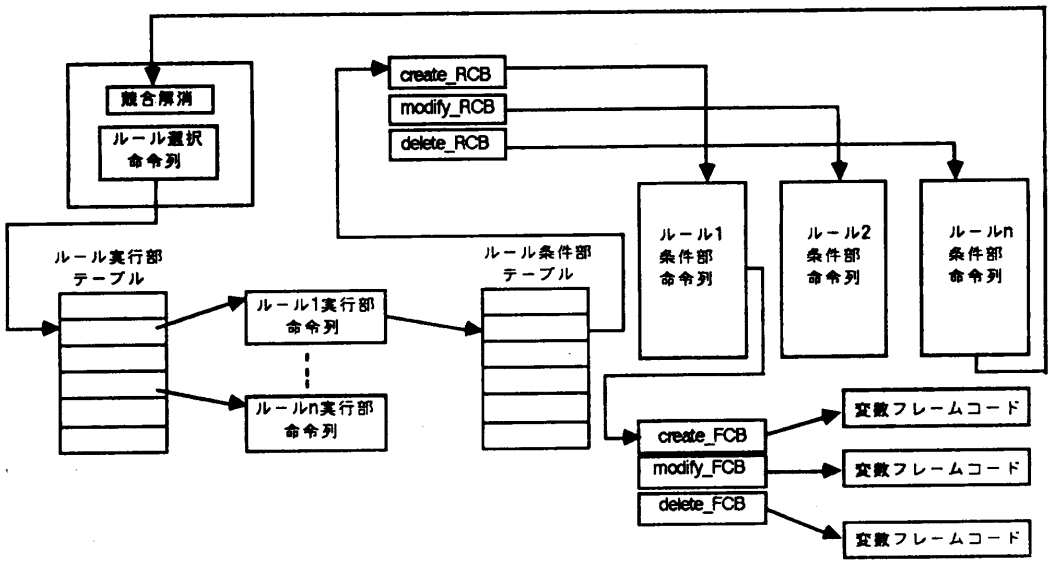

図 9 コンパイル型推論の全体構成
Fig. 9 Processing overview for a compiled-type production system.

図5に示したごとく中間結果と競合集合を更新し，新たなインスタンシエーションを中間結果と競合集合へ追加する．この処理は前述したとおりである．以上の処理を実行した後，NGR を参照することにより競合集合命令列へ再び戻ることができる．ここでもし競合集合が空集合であれば推論終了であるが，そうでなければ再びインスタンシエーションを競合集合から選択し推論を繰り返すことができる．以上，コンパイル型前向き推論の基本的な動作を説明したが，最後にルール条件節において，参照すべきフレームの探索範囲をできるだけ小さくすることにより，フレーム探索の高速化を実現する方式について述べる．また定数フレーム条件節では，唯一つのフレームのみを参照すれば良いので探索上の問題はない．一方，変数フレーム条件節は複数のフレームを参照しなければならないため，その探索範囲を絞り込むことが非常に重要である．その1つの方法として，変数フレーム条件節には，クラスフレーム名とスロット名が共に記述されるケースが非常に多い（表2参照）という特徴を用いた探索技法が考えられる．例としてクラス f に属するインスタンスフレームとして fa, fb, fc が存在し，一方ルールは，その条件節にクラス名 f とスロット名 slot 1 が指定されているケースについて考えてみる．この時，ルールは上記3つのインスタンスフレームを参照する必要がある．本探索技法は，図8に示したごとく，スロット名とクラスフレーム名を索引(-slot1-f)とし，そのアドレスにフレーム制御命令列（create-FCB -slot 1-fa 命令，modidy-FCB -slot 1-fb 命令，delete-FCB -slot 1-fc 命令）を配置し，さらにそのアドレス -slot 1-f(i)，($i=a,b,c$) には各インスタンスフレームの slot 1 の変数フレームコードアドレスを格納しておくことにより，ルールは，必要最小限のフレーム探索を実現できるようになる．また，ルール条件節にクラス名 f の記述がなく，スロット名 slot 1 のみ記述されている場合には，同様にスロット名を索引（-slot 1）として，そのアドレスにフレーム制御命令列を配置し，各制御命令から slot 1 を有するすべてのフレームの変数フレームコードへ分岐するようにすれば，参照すべきすべてのフレームの探索を実現できる．このように，本探索技法はクラス名とインスタンス名がルールに記述されている場合に，その探索範囲を大幅に絞り込むことが可能である．しかしながら，フレーム数が多くなると，個々のクラス名とスロット名の組合せに応じてフレーム制御命令を必要とするため，命令コードサイズが増大するという問題や，コンパイラが静的に探索順序を決定するので，探索順序が固定されるという問題が生ずる．このため命令コードサイズの問題に対しては，クラス名とスロット名をハッシュし，そのハッシュ値を索引として，そのアドレスに同一ハッシュ値を持つ複数のクラスとスロットに係るフレーム制御命令列を配置することにより，フレーム制御命令列を削減できる．一方，探索順序が固定される点については，ルールに記述されたクラス名とスロット名に応じて，コンパイラが最適な

フレームの探索経路を生成するので，探索順序を固定してもかなり効率良く探索を実現できるものと思われる．

3.4 フレームの生成，削除方式

本節ではインスタンスフレームの生成，削除の実現方式について述べる．一般にプロダクションシステムは，推論中にインスタンスフレームを生成したり，削除することができる[1]．しかしながら本論文で述べたコンパイル型推論方式は，高速化のためにフレームを命令列に変換することを特徴としており，その結果，フレームの生成処理はフレーム命令列を推論中に追加することを意味する．さらに追加されたインスタンスフレームをルールから参照できるようにするには，フレームコントロール命令列も変更しなければならない．一方，インスタンスフレームの削除処理も，フレーム命令列とフレームコントロール命令列を削除することを意味しており，プログラムの信頼性に問題がある．またオペレーティングシステムの観点から見ても，複数のユーザ間でフレーム命令コードを共有不可となるため個々のユーザごとに命令コードを割り当てることになり，大容量のメモリが必要になるという欠点がある．そこで我々は，下記方針にて上記欠点を解決することとした．

- 実行時に生成するインスタンスフレームの命令列は，クラスフレームの命令列と共有化する．
- フレームコントロール命令列は，コンパイル時に存在するフレームを対象とする静的フレームコントロール命令列と実行時に生成されるインスタンスフレームを対象とする動的フレームコントロール命令列の2種を用意し，分岐先アドレスを命令コードから分離してデータ領域に配置する．

まずインスタンスフレーム生成問題に関しては，生成時に必ずクラスフレームが指定され，かつクラスフレームは削除されないという性質を利用してインスタンスフレームとクラスフレームの命令コードを共有化することとした．このため，クラスフレームの命令列は，**図 10** に示すように図8のインスタンスフレーム命令列と異なる．すなわち，インスタンスフレームの命令列ではフレーム名 (#fa)，タイムタグアドレス (time_adr)，スロットアドレス (slot 1_adr) を命令内で直接指定されるのに対し，クラスフレームの命令列ではフレーム名 (#disp1 (R4))，タイムタグアドレス (R4)，スロットアドレス (#disp 2 (R4)) をレジスタ R4 の相対アドレスで参照する．これによりクラスフレームの命令列は，R4 を変更することでインスタンスフレームを参照できる．例えば，図 10 の例に示すインスタンスフレーム fb の生成では，クラスフレーム f のデータをコピーして fb のデータを作成し，そのタイムタグアドレスをレジスタ R4 へ設定し

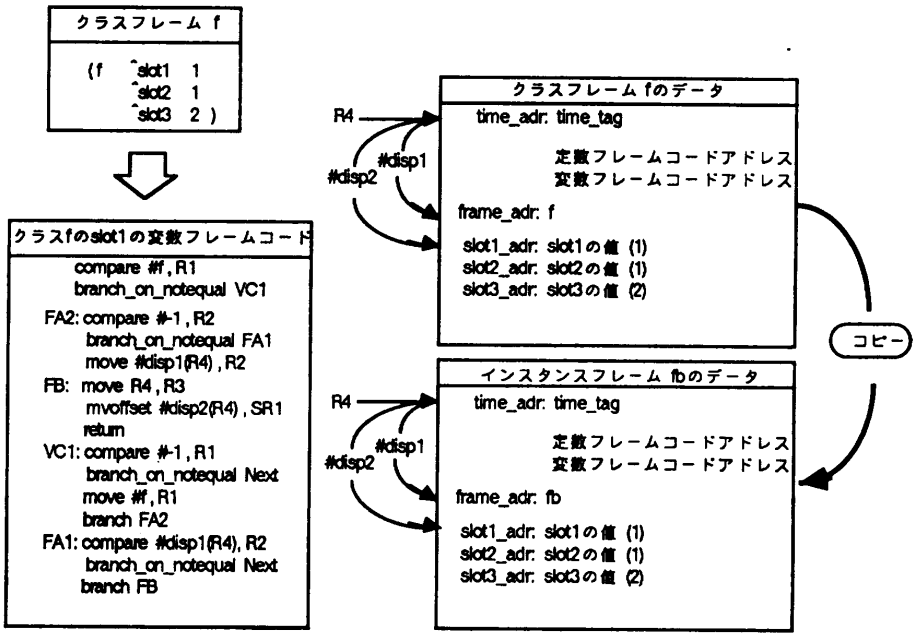

図 10 クラスフレーム f の命令コードとインスタンスフレームの生成データ
Fig. 10 Instruction codes for class frame f and created instance frame fb in inferencing.

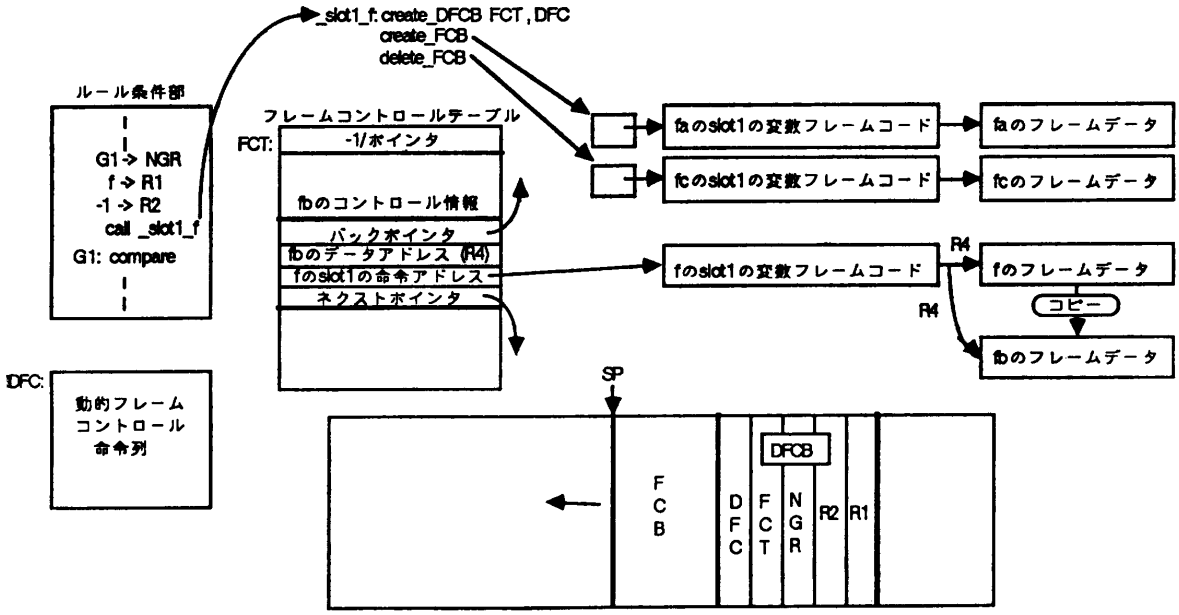

図 11 フレームコントロールテーブルと動的フレームコントロール命令列
Fig. 11 Frame control table and dynamic frame control instructions.

た後にfのクラスフレーム命令列へ分岐すれば，fbを参照可能であることを示している．また，これにより前節で述べたルールとフレームのインタフェース規約も満足することができる．次に，動的フレームコントロール命令列を**図 11** を用いて説明する．ここでルール条件部命令列は，図8に示した変数フレーム条件節であり，fはクラスフレームである．また，faとfcはインスタンスフレームであり，fbは実行時にfから生成されたインスタンスフレームである．この例からわかるようにフレーム生成時には，fをコピーしてfbのフレームデータを作成するだけでなく，フレームコントロールテーブル中にfbのコントロール情報を生成する．この結果，動的フレームコントロール命令列は，コントロール情報を用いてfbを参照できるが以下説明する．まず，ルール条件部からcall命令が実行されるとcreate_DFCB命令によってスタック上にダイナミックフレームコントロールブロックDFCBが生成される．ここで，FCTはfbのコントロール情報アドレスであり，DFCは動的フレームコントロール命令列アドレスである．次にcreate_FCB命令とdelete_FCB命令によりFCBを生成，削除することによってそれぞれfa，fcが実行される．その後DFCBのDFCをスタックから読み出すことによって動的フレームコントロール命令列を実行することができる．この命令列は，DFCBのFCTを読み出してfbのコントロール情報内のデータアドレスをR4へ設定し，ネクストポインタをDFCBのFCTへ格納した後，fbのコントロール情報を用いてfのslot1の変数フレームコードアドレスへ分岐することができる．このように，動的フレームコントロール命令列とフレームコントロール情報を用いてDFCBを生成，更新，削除することにより，生成されたインスタンスフレームを参照することができる．一方，フレームの削除問題は，フレームデータの削除とフレームコードアドレスの変更を行えば良い．例えば，図11のfaやfcを削除するには，フレームコードアドレスを保持するデータ領域を書き替えることで実現できる．一方，実行時に生成されたフレームを削除するには，そのコントロール情報をフレームコントロールテーブルから削除し，他のコントロール情報をチェインで結ぶことで実現できる．コントロール情報内のバックポインタとネクストポインタは，このチェインを結ぶ時に使用する．

以上により，インスタンスフレームの生成と削除は可能となったが前章で述べたように，インスタンスフレームが生成されるケースはごくまれである．このため，下記方法により，さらにフレーム参照を高速化できる．すなわち，フレームコントロールテーブル中にフレームの生成有無を示すフラグを保持し，フレームを生成した時に，そのフラグをON状態とすること，

さらに，create_DFCB 命令の処理仕様を ON の時には DFCB を生成し，OFF の時には DFCB を生成しないようにすることにより，通常の OFF 状態におけるフレーム参照を高速化できる．このようにコンパイル型推論方式は，フレームの生成，削除についても効率良く処理することができるため，高速推論に適している．しかし，本方式は，クラスフレームを生成・削除可能なプロダクションシステムには，適用できない．ただし，クラスを動的に生成・削除可能にするとプログラムのデバッグがしづらい等の問題が生じるため，多くのプロダクションシステムでは，その生成・削除を許していない．この意味で，本方式は一般性を失っていないと思われる．

4. 性能評価

本章では，フレームの命令列表現に基づくコンパイル型推論方式とデータ表現に基づくインタプリタ型推論方式の性能比較を行う．両者の推論アルゴリズムとしては，多量の中間結果を保持するものから全く保持しないものまでいろいろ存在するが，ここでは中間結果を保持する RETE アルゴリズムを採用した．またコンパイル型の性能評価は，3章で述べた方式に基づいてハンドコンパイルし，そのマシンサイクル数をカウントすることにより行い，一方，インタプリタ型はプロトタイプを作成し，HITAC E-7700 H (メモリ 128 MB) 上でその実行時間を実測することにより評価した．

4.1 フレーム参照の性能評価

図 12 にコンパイル型 (CMP) とインタプリタ型 (INT) のスロット参照性能(a)，スロット値演算性能(b)，ルール実行部におけるスロット値更新性能(c)を示す．この結果，コンパイル型は，インタプリタ型に比較して，スロットの参照は 8.5 倍，スロット値の演算は 25 倍，スロット値更新は 22 倍高速である．この原因は，フレームの命令表現の効果とルール/フレーム間のインタフェースをレジスタ渡しにしたためと考えられる．ただし，コンパイル型のフレーム参照時間は，変数フレームコードの実行時間と create_DFCB 命令による DFCB 生成時間と create_FCB 命令による FCB 生成時間を含んでおり，フレーム参照が最も遅くなるケースを用いて評価した．一方，定数フレーム条件節からのフレーム参照時間は，定数フレームコードの実行時間とスタック上に DFCB を生成する create_DFCB 命令の実行時間であるため 12.5 倍高速であり，さらにフレームが生成されていない場合には create_DFCB 命令は，DFCB をスタックに生成しないため，14.5 倍高速である．以上の結果，コンパイル型はルールにおけるスロット値判定演算を 25 倍も高速化できることがわかった．このため，フレームの参照もそれに見合った高速化をする必要があり，本論文で提案したフレームの命令列表現方式は，その1つの解と思われる．

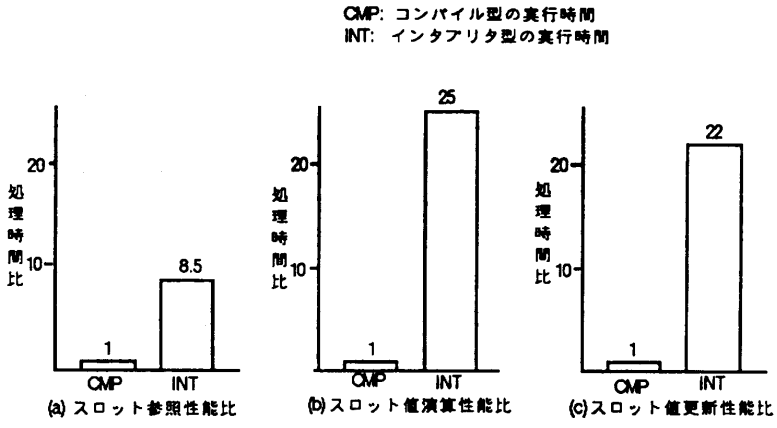

図 12 フレームの命令列表現に基づくコンパイル型推論方式の性能
Fig. 12 Performance of compiled-type inference processing based on frame instructions.

4.2 定数フレーム条件と定数フレームコードの性能

本節では，条件節数と参照するフレーム数の増加に伴い，パターンマッチ性能が，インタプリタ型とコンパイル型でどの程度異なるか調べる．図 13 は，m 個の条件節を持つルールである．このルールを用いて性能比較を行った．その結果を図 14 に示す．この図からわかるように，照合のための前/後処理に要する時間は，インタプリタ型とコンパイル型では，各々 1,970 mc と 134 mc であり，1つの条件判定とフレーム参照に要する時間は，各々 485 mc と 34 mc である．また，$m=3$ の時の性能比は，14.5 倍であり，コンパイル型は非常に高速であることがわかる．

4.3 変数フレーム条件と変数フレームコードの性能

図 15 のルールとフレームを用いてコンパイル型とインタプリタ

```
if ( (fr 1 ^slot 1=1)
     (fr 2 ^slot 2=2)
          ⋮
     (fr m ^slot m=m) )
then……
```

図 13 m個の定数フレーム条件節を持つルール
Fig. 13 A benchmark rule with m constant frame conditions.

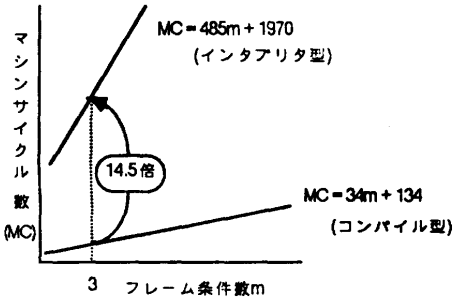

図 14 m個のフレーム条件数を持つルールの性能
Fig. 14 Pattern match performance of a rule with m constant conditions.

```
(rule1 if
 (?X ^class=fb
     ^d->?YX)
 (?Y ^class=fa                    (class fa ^super_class system
     ^b<10                              ^b          1    )
     ^frame_name->?Z)
then                              (class fb ^super_class system
 (assign fa1 ^b+1-> ^b)                 ^d          1
                                        ^e          2    )
(rule2 if
 (?X ^class=fa
     ^b<10      )                 (fa1    ^class    fa   )
 (?Y ^class=fb                     (fb1    ^class    fb   )
     ^e=2                          (fb2    ^class    fb   )
     ^d->?YX    )
then
 (assign fb1 ^e+1-> ^e)
```

図 15 ルールとフレーム
Fig. 15 Rule and frame.

型の性能比較を行った．ここでルール1と2の条件節は，すべて変数フレーム条件節である（変数？X，？Yが変数フレームであることを示している）．**図 16** に認知行動サイクルの実行時間を示す．RETE アルゴリズムに基づくコンパイル型は，インタプリタ型に比較して 19.5 倍高速であるが，その高速化要因は，FCB と RCB に基づくフレーム参照/ルール実行制御が効率的であること，またルール条件部における照合判定処理が 10〜20 命令で済んでいること等による．しかしながら，RETE アルゴリズムでは，フレームが更新されると各条件部ごとの中間結果（αメモリ[2]）ばかりでなく中間結果の組合せ（βメモリ[2]）も更新しなければならないため，メモリ管理のオーバヘッドが大きい．そこで，各条件節には中間結果を保持するが，複数条件節に渡る中間結果は保持しないアルゴリズム（TREAT[9]）に基づくコンパイル型の推論性能を評価した．その結果，ルール1，2共に，第1条件節と第2条件節の join 操作を省略できたこと（すなわち，メモリ管理操作量を削減できたこと）により，RETE に基づくコンパイル型に比較して，ルール条件部処理に要する命令数が 1,626 命令から 1,253 命令に減少し，認知行動サイクル全体では 1.23 倍さらに高速化できた．

5. おわりに

ルールとフレームを共に命令列に変換して推論を行うコンパイル型推論方式を提案した．本方式の特徴は，コンパイラがレジスタを効率良く使用できること，フレームのスロットを高速に参照できることであり，しかもフレームの生成，更新，削除を実現できる

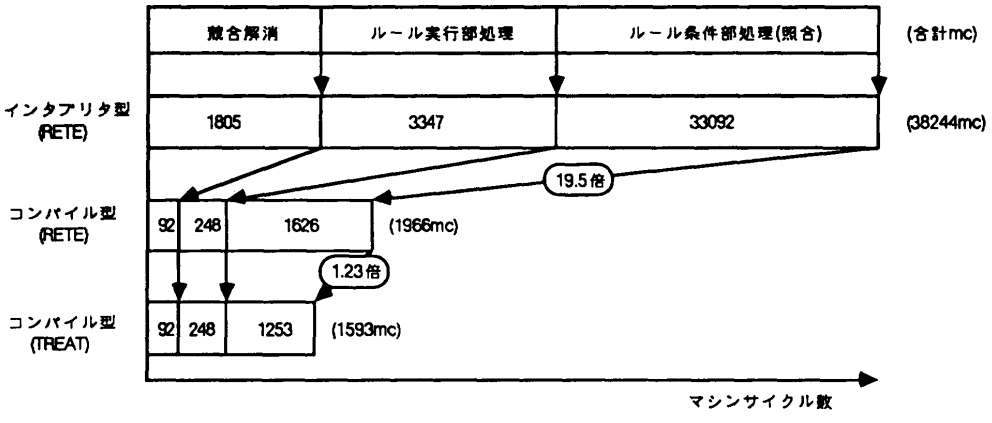

図 16 変数フレーム条件節を持つルールの実行時間
Fig. 16 Execution time of rules with variable frame conditions.

点にある．本方式の効果を簡単なルールを用いて評価した結果，コンパイル型のプロダクションシステムは，インタプリタ型に比較して 14〜20 倍高速化できることを明らかにした．しかし，さらに高速化を実現するには，RETE や TREAT に代わり，照合判定回数とメモリ管理オーバヘッドを評価指標とする新しい推論アルゴリズムが必要である．

参考文献

1) Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem, *AI*, Vol. 19, pp. 17-37 (1982).
2) 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).
3) 中村：プロダクションシステム ARCH-2 の処理系概要と性能評価，第 1 回人工知能学会全国大会論文集，pp. 193-196 (1987).
4) Allen, E.: YAPS: A Production System Meets Objects, *AAAI '83*, pp. 5-7 (1983).
5) 島田，黒沢，平山：IPP による汎用エキスパートシステムの高速化技術，情報処理学会計算機アーキテクチャ研究会資料，89-ARC-74, pp. 35-42 (1989).
6) 黒沢，島田，平山：汎用エキスパートシステムの高速化技術(I)開発構想，第 39 回情報処理学会全国大会論文集，pp. 462-463 (1989. 10).
7) 島田，黒沢，平山：汎用エキスパートシステムの高速化技術(II)推論方式，第 39 回情報処理学会全国大会論文集，pp. 464-465 (1989. 10).
8) 平山，島田，黒沢：汎用エキスパートシステムの高速化技術(III)性能評価と分析，第 39 回情報処理学会全国大会論文集，pp. 466-467 (1989. 10).
9) Miranker, D. P.: TREAT: A Better Match Algorithm for AI Production Systems, *AAAI '87*, pp. 42-47 (1987).

(平成元年 12 月 9 日受付)
(平成 2 年 6 月 4 日採録)

黒沢 憲一 (正会員)

昭和 28 年生．昭和 55 年東北大学大学院工学研究科修士課程情報工学修了．同年(株)日立製作所入社．知識処理計算機の命令アーキテクチャ，PROLOG 言語の最適コンパイラの研究に従事．人工知能マシンアーキテクチャ，並列処理に興味をもつ．現在，同社日立研究所第 8 部研究員．電子情報通信学会会員．

島田 優 (正会員)

昭和 35 年生．昭和 59 年東北大学工学部通信工学科卒業．昭和 61 年同大学院修士課程情報工学修了．同年(株)日立製作所日立研究所に入社，主としてコンパイラの研究に従事．分散環境に興味を持つ．