記　号　処　理　56－5
計算機アーキテクチャ　84－5
（１９９０．　９．　２８）

機能メモリに基づくProlog処理系の構成

坂井雄介、小原清弘、五十嵐智、小谷善行、阿刀田央一、斎藤延男
東京農工大学工学部

パターン連想メモリ（Pattern Associative Memory: PAM）や可逆メモリ（REversible Memory: REM）を用いたProlog処理系を試作し、ベンチマークプログラムを走らせた評価について述べる。PAM をProlog処理系へ適用する際に行った検討事項についても述べる。PAMはホーン節を格納し候補節の連想検索を実現する。全ての引数にClause indexing を行ったのと同等な節を出力するため、候補節を絞り込み単一化失敗の回数を減少させる効果が確認された。REMは記憶内容の回復機能を持った機能メモリであり、REM上に変数スタックを取ることにより変数の未束縛化作業を高速化・自動化する。これによりバックトラック時間が約7.5%となり、大幅な高速化が認められた。

Prolog processor based on ASIC memories

Yusuke Sakai　Kiyohiro Obara　Satoshi Igarashi　Yoshiyuki Kotani　Oichi Atoda　Nobuo Saito

Faculty of Technology, Tokyo University of Agriculture and Technology
2-24-16 Naka-chou, Koganei-shi, Tokyo 184, Japan

We describe Prolog processor to which two Application Specific Integrated Circuit (ASIC) memories are applied. One is Pattern Associative Memory(PAM) and another is REversible Memory(REM). Horn clauses are stored in PAM. PAM decreases the number of unification failures, owing to search such clauses as apply clause indexing to each argument. REM is assigned to trail stack. While backtrack, it can rapidly and automatically unbind instances. In this paper, we describe some examinations gluing PAM to Prolog processor, and several evaluation of REM & PAM. As results of evaluation, the number of unification is decreased using PAM. Backtrack time decreases to about 7.5% using REM.

１．はじめに

Prologはその言語的特徴から知識処理や人工知能の分野で広く用いられている。しかしその反面、従来の手続き型言語（C,FORTRAN等）に比べて処理速度は 10分の1（汎用計算機の場合）程度である。Prolog において、その処理の中で大きな負荷となるのは、

(1)メモリからの節の検索

(2)単一化におけるゴールと節のマッチング

(3)バックトラック

の各処理である。これらの処理は、特に複雑ではない。しかし、まとまった処理を行うまでに、CPU とメモリ間で多量のデータトラフィックが生じ、それらが系全体の実行時間を支配してしまう。

記憶に関する処理に対し、メモリに制御回路を付加し、ある程度のまとまった処理をメモリ側で行わせることが、高速化の一方策として考えられる。特に、候補節検索についてはメモリ内部で検索処理を行い、メモリープロセッサ間の通信データ量を減らすことが可能である。また、バックトラックについてもプロセッサ側で管理するのではなく、メモリ側にバックトラック機能を持たせることにより、プロセッサの負荷を減らすことが可能である。

我々はこのような機能メモリを指向した論理型マシンアーキテクチャを研究しており、これまでパターン連想メモリ（Pattern Associative Memory：以下PAM）[1]と可逆メモリ（Reversible Memory：以下REM）[2]を提案し、それぞれ単一化候補節検索とバックトラック処理の高速化を試みている。REM・PAMの機能については文献[3][4]でも報告している。本稿２章では PAMのProlog処理系への適用に関する議論とProlog処理系の構成を述べる。３章では、REM・PAMをProlog処理系へ適用した場合の、ベンチマーク・プログラムによる測定および評価について述べる。

２．ＰＡＭを用いたProlog処理系

２．１．ＰＡＭの効果

PAMは候補節の連想検索を実現する。PAMに格納されるデータは、節すなわちPrologのプログラムそのものである。PAM はCPUにI/Oとして結合し、アドレスの代わりにゴールを受け取り、それとパターンマッチ可能な節を自身の中から連想検索しCPU へ返す。パターンマッチ規則は、Prologの単一化に沿ったものである。PAMを用いることにより、Prolog 処理系の候補節検索を処理系から分離できる。したがって、検索時に CPUとメモリ間で発生する大量のデータトラフィックを回避できる。さらに、PAMとCPUが並列に処理を行うことにより系全体の効率化、高速化が図れる。この手法は、特に同一述語名の多い知識処理などの応用に有効である。

Prologでの節検索の効率化手法に、clause indexingがある。PAMは、いわばすべての引数に対して clause indexingを行ったのと同等な節を出力する。これにより、失敗単一化の数を減少させることが望める。また、節の管理が処理系からPAM へ移るため、処理系が簡素化される利点もある。PAMについての詳細は、文献[1][4]を参照されたい。

２．２．ＰＡＭのProlog処理系への適用

PAMに対応した処理系を試作するにあたり、PAMと処理系との整合性の観点から検討を行った事項について述べる。

(1) 共有変数の一貫性チェック

PAM は、節内部の共有変数の一貫性チェックは行わない。したがって、同一変数名に違う値がマッチする節を候補節として出力する可能性がある。現在、PAM内での連想検索と共存できる適切な一貫性チェックアルゴリズムを得ていないため、共有変数の一貫性チェックは、ホスト側で行うようにした。

(2) ハッシュテーブル数

いくつかの典型的なPrologプログラムについて静的解析した結果、述語の引数の数は平均３個であった。このため、述語部のハッシングをするテーブルを一つ加え、ハッシュテーブル数は４に設定した。しかし、試作したハードウェアの制約から現在は３となっている。

(3) 項目数がハッシュテーブル数を越えたときの項目の割当て

引数の数がハッシュテーブル数を越えた場合、その引数は、最後のハッシュテーブル（現在は３）に割り当てる。これにより、並列構成要素を容易に増やすことができる。

(4) 複合項の格納について

本来、PAM には節の情報を一括して格納すべきである。しかし、PAM 内のメモリ容量およびホストCPU と PAM 間の通信容量の制約から、複合項はPAM に格納しないこととした。すなわち、節中の項目のレベル１まで（複合項以外）をPAM に、レベル２以上（複合項）をホストプロセッサのローカルメモリに格納することとした。したがって、複合項はハッシュの対象から除外し、複合項が出現する度にホストプロセッサがマッチングを行う。

２．３．処理系

REM・PAMを組み込んだ処理系の構成および動作について説明する。インタプリタはＣ言語で記述した。

REM上に変数を格納し、PAMには複合項以外の節を格納する。

この処理系では、インスタンスは構造複写法で表現され、データ表現は疑似レコード形式を採用している。スタックは、制御情報を格納する制御スタック、実行環境の管理情報を格納する変数スタック、変数を未束縛に戻すためのトレイルスタック、節とインスタンスのコピーを格納するコピースタックの４本である。

次に制御スタックによる実行制御を示す。

① 制御スタックが空の場合には、trueを返す。

② 制御スタックのトップに対応する述語（ゴールリテラル）をPAMとの共有メモリに格納する。

③ PAM は②の述語を述語項と各引数項に分け、各ハッシュテーブルからマッチする項目を探し出し、一致したものを共有メモリに格納する。無ければ、failを返す。

④ 単一化を行う。単一化失敗なら、⑦へ飛ぶ。

⑤ 共有メモリ内の節が規則節ならば本体をコピースタックに積む。このとき変数はそのインスタンスをコピースタックに格納する。制御スタックに新たな選択点を積む。

⑥ ①から⑧までを再帰的に呼び出す。trueが返ったらtrueを返す。failが返ったら、⑤で積んだ情報をスタックから除く。

⑦ バックトラックを行う。

⑧ コピースタックに積まれている述語の次の候補を探す。無ければ、failを返す。見つかれば、④へ飛ぶ。

インスタンスの表現に構造複写法を使ったのは、規則節を単一化した時、⑤のようにコピースタックにインスタンスへのポインタの代わりにインスタンスを格納するだけで、その規則節の本体部をそのままゴールリテラルとしてPAM との共有メモリに転送することができるからである。

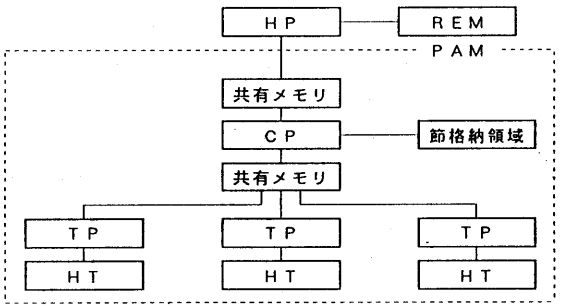
また、深いバックトラックが起きた場合、Prologでは必ず戻った選択点のゴールの次候補節の検索を行う。このことから以前単一化に使用したゴールをPAM 内のスタックに保存しておき、それを深いバックトラック時に用いることにより次候補節検索の再開を容易に行う方法が考えられる。しかしこれらの情報は処理系が管理すべき情報であるため、PAM にはこの方法は取り入れていない。

３．測定・評価

３．１．ＲＥＭの測定・評価

試作機の構成を図１に示す。ホストプロセッサは、MC68000をCPUとしたシングルボードコンピュータであり、Prologインタプリタが置かれる。クロック速度は、8MHzであり0.5MByteのRAMを持つ。

REM は書換えにより失われた記憶内容の回復が可能な機能メモリである。REM 上に変数を取ることにより、バックトラック時の変数の値の未束縛化作業がREM により自律的に行われる。



ＨＰ：Host Processor.　　ＴＰ：Table Processor
ＣＰ：Control Processor.　　ＨＴ：Hash Table
ＰＥ：CPU MC68000(8MHz), Memory 0.5Mbyte

図１　試作機の構成

表１　単一化回数とバックトラック回数[REM]

プログラム	（１）	（２）	（３）	（４）
１－１	107	31	208	6.7
１－２	152	76	748	9.8
２	32	28	23	0.8
３	98924	98908	207163	2.1

但し、（１）単一化回数　　（２）バックトラック回数
（３）未束縛に戻る変数の総数　（４）（３）／（２）

表２　実行時間の比較[REM]

プログラム	実行時間（全体）単位：秒		比(%)	実行時間(ﾊﾞｯｸﾄﾗｯｸ)単位：ﾏｲｸﾛ秒		比(%)	実行時間短縮量(1)単位：秒	バックトラック時間短縮量(2)単位：秒	(1)-(2)単位：秒
	REM使用	REM未使用		REM使用	REM未使用				
１－１	0.5636	0.5865	96.10	336	10509	3.20	0.0229	0.0102	0.0127
１－２	0.7345	0.7900	92.97	822	37246	2.21	0.0555	0.0364	0.0191
２	0.0961	0.0975	98.56	302	1495	20.20	0.0014	0.0012	0.0002
３	195.7120	207.7815	94.19	1074852	11379620	9.44	12.0695	10.3048	1.7647

比：使用／未使用

評価に使用するプログラムは次の三つである。なお、この測定にはPAMは使用していない。

括弧内はそれぞれ節の数とその性質を表す。

１.クイックソート［１０個降順、昇順］
（5、おもにＬＩＳＴ項）

２.意味ネットワーク（535、おもに定数項）

３.機械翻訳（790、おもに複合項）

表１に単一化回数とバックトラック回数を、表２に実行時間を示す。

表１、表２より、REMを使用することにより、１回のバックトラックで未束縛に戻る変数の数が多いほど実行時間は短縮されることが確認できた。これは、REMの利用により、変数の未束縛化処理の時間が不要になったからである。

現在REM はTTL-ICで実現されている。平均アクセス時間はREAD が354ns、WRITEが167nsである。処理系が、REM へバックトラックコマンドを与えるに要する時間は10.5μsである。REMは、バックトラックのコマンドを受け取った後、CPUと独立に変数の未束縛化を行う。この試作機上で評価を行った結果、変数を多く含むプログラムほどREM の効果は大きく、プログラム２のようなバックトラック時に未束縛にする変数の数が平均で１以下の場合でも速くなっている。他のプログラムを含めた平均では、バックトラック時間は約7.5%に短縮された。

全体の実行時間の短縮量はバックトラック実行時間の短縮量より多いという結果がでた。これは、代入変数のトレイル処理の時間である。このようにREM の利用により、バックトラック処理の処理系からの分離・高速化に加え、それに備える処理をも省略でき、処理系の簡素化に大きく貢献する。

３.２.ＰＡＭの測定・評価

PAM のハードウェア構成において、CP及びTPは専用のシーケンサによる構成が最適である。しかし、開発環境や各種コストの点から、クロック速度8MHzのMC68000 とソフトウェアでエミュレートしている。ホストプロセッサーCP間及びCP－TP間は4KByteの共有メモリを通して結合している。各HTの大きさは、0.5MByteである。HTの容量は容易に拡張可能である。

評価プログラムには次の四つを用いた。なお、この測定にはREMは使用していない。

括弧内は節の数と性質を表す。

１.クイックソート（5、おもに複合項）

２.Ｎクイーン（15、おもに複合項）

３.意味ネットワーク（535、おもに定数項）

４.機械翻訳（790、おもに複合項）

比較のために、PAM を使用するインタプリタと使用しないインタプリタを用いて、単一化・単一化失敗・バックトラック・組込み述語実行の回数を測定した。プログラム２の実行回数を表３に示す。

表３　Ｎクイーンでの実行回数（最初の解を見つけるまで）[PAM]

N	ＰＡＭ使用（回）				ＰＡＭ未使用（回）			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
4	320	57	285	54	430	154	285	57
6	1114	188	1063	183	1672	621	210	1205
8	35039	5063	36288	6375	52294	18581	40996	7346

但し、(1)単一化回数回数　　(2)単一化失敗回数回数
　　(3)バックトラック回数　　(4)組込み述語実行回数

表４　単一化回数と単一化失敗回数[PAM]

プログラム	単一化回数（平均）			単一化失敗回数（平均）		
	使用	未使用	比	使用	未使用	比
Ｑソート	127	166	0.73	0	39	0.0
Ｎクイーン	8490	12657	0.69	1250	4522	0.31
意味ネット	3	65	0.04	0	62	0.0
機械翻訳	111407	132945	0.85	109931	131469	0.85

比：使用／未使用

表３を見ると、いずれのプログラムも単一化失敗回数が少ない分だけ単一化回数が減少している。これにより、PAM の利用による単一化回数の抑制効果を確認した。

また、それにつれてバックトラック回数や組込み述語の実行回数も減少している。すなわち、PAM の利用により、単一化回数の抑制のみならず、バックトラック回数や組込み述語の実行回数をも抑制することができ、それらに要した時間をも省くことができる。

PAM の利用による主な効果は、前述のように候補節の絞込み及び候補節検索の高速化である。この効果をプログラム別に見たのが、表４のPAMの使用/未使用による平均の単一化回数と単一化失敗回数の比である。

これによると、PAM の使用により最も効果が表れたのは、意味ネットワークである。このプログラムは、ほとんどの節の頭部の引数が定数項である。従って、PAM の出力した節はすべて単一化が成功している。逆に機械翻訳では、データの表現手法上複合項が多い。このため、PAM による候補節の絞込みの効果があまり表れていない。このように、頭部の引数に変数や構造体が多い節などでは絞込みの効果が表れにくいが、データベースのような定数引数の多い節に対しては、その効果は絶大である。

この測定に用いられたPrologインタプリタは、候補節検索部分をPAMに置き換えただけの構成であり、PAM と処理系は並列に動作しない。また、汎用プロセッサという低速なエミュレータによる実現などの理由により、実行時間の比較は意味をなさない。強いてこの試作機で実行時間を調べてみると、PAM を使うことで１回の単一化に約4.3倍かかっている。しかし、シーケンサを用いたPAM の専用ハードウェア化などによりこのボーダーラインは下げることができる。また、同一述語名の節が多い大規模なプログラムでは、PAMの利用効果が如実に現れるであろう。

現在のPAM では複合項に対するパターンマッチを行っていない。従って、機械翻訳のように複合項を多く含む節に対しては、PAM を用いた場合でも候補節の絞込みが効果的に行われない。これへの対応も課題の一つである。

４.結論

本研究では、論理型言語の実行の高速化を実現するアプローチとして機能メモリによる方式を試み、論理型言語を扱う際のボトルネックであった候補節検索の絞込み及び高速化を実現する、PAM のPrologへの整合を検討した。またREM・PAMについてそれぞれ測定・評価を行った。

REM を使うことにより、平均でバックトラック時間が約7.5%、全体の実行時間が約95%になることが分かった。PAM ではほとんどのプログラムで単一化回数が減少するのを確認でき、節の数が多い大規模なプログラムでは効果が期待できる。

今後の課題としては、REM についてはより実用的なものとするためのカスタムIC化が考えられる。PAM については、単一化回数を減らし、実行時間を速くするために複合項等の取り扱いを再度検討する必要がある。さらに制御アルゴリズムにも改良の余地がある。また今回は汎用プロセッサでPAM をエミュレートしたが、より実用的なものとするため、高速なRISCチップをコントローラとして組み込んだ専用ハードウェアを現在構築中である。

PAM・REMをそれぞれ単独でProlog処理系へ組み込み、評価を行うことにより、机上では発見できなかった問題点や多くの知見を得た。これを、現在構築中の PAM とREM 両方を含む推論マシンの設計へフィードバックさせることにより、より洗練された高性能のシステムの構築が可能になると思われる。

謝辞

本研究を共同で進めてくださった曽我和美（現、日立製作所（株））、Prologインタプリタを提供してくださった内原亜紀（現、松下電器産業（株））・駒形真（現、（株）富士通）の各氏に感謝します。

参考文献

[1]安田 ほか：「パターン連想メモリ」とその論理言語処理系への応用，信学論(D)，Vol.J71-D，No.9，pp.1614-1622 (1988-9)

[2]川藤 ほか：可逆メモリ，信学論(D)，J-71D，No.9，pp.1614-1622(1987)

[3]小原 ほか：可逆メモリのProlog変数管理への応用，信学論(D-I)，Vol.J72-D-I，No.2，pp.136-139 (1989-2)

[4]小原 ほか：機能メモリに基ずく推論機械のアーキテクチャ，情報処理学会記号処理研究会資料56-4 (1990)