

最適化PROLOGインタプリタの性能評価

7Q-5

碓崎賢一 打浪清一
九州工業大学 情報工学部

1 はじめに

PROLOG は、プログラムの宣言的記述が可能で、単一化と後戻りを基本とした柔軟で強力な処理能力を備えているという特徴を持つが、手続き的な言語と比較して、処理速度やメモリ効率が低いという問題点をかかえている。この問題はインタプリタにおいて顕著であり、プロトタイピングを効率よく行えることやデータをプログラムとして実行できるという PROLOG の特徴を損ねる要因になっている。

本報告では、WAM^[1]と基本的に同じ実現方式を採用した上で、さらにいくつかの新たな最適化手法を導入し、処理速度とメモリ効率を向上させた最適化 PROLOG インタプリタ^[2]の最適化方式の概略と性能評価を示す。

2 最適化方式

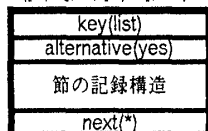
2.1 インデクシング

インタプリタでは、実行時にプログラムが変更される可能性があるために、インデクシングを効率よく行うことが困難とされている。本インタプリタでは、処理時間が少なくすむインデクシング情報の作成法と、節を選択した時点で代替節の有無を検出する方式を導入することでインデクシングを実現している。

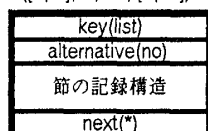
2.1.1 インデクシング情報の作成

インデクシング情報として、述語の第1引数のデータタイプを利用して作成したインデックスキーと、そのインデックスキーをもとに調べた代替節の有無の2種類の情報を節の登録時に作成し記録する方式^[2]を導入している。この処理は、各節に対する操作が単純なインデックスキーの比較と代替節の有無の記録のみなので、データベースに節を格納する処理時間より十分短い時間で終わる。

partition([X|L], Y, [X|L1], L2) :- X=<Y, partition(L, Y, L1, L2).



partition([X|L], Y, L1, [X|L2]) :- partition(L, Y, L1, L2).



key(?)
インデックスキー
alternative(?)
代替節の有無

partition([], ..., [], []).

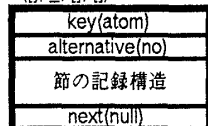


図1 インデックスキーと代替節の有無の記録

インデックスキーと代替節の有無が記録された節の記録構造を図1に示す。この例では、第1, 第2節ではリストが、第3節ではアトムがインデックスキーとして記録されている。したがって、ゴールの第1引数が nil の場合には、インデックスキーが合致しない第1, 第2節を無視して、第3節を直接選択でき高速な処理が可能になっている。また、第2節が選択され失敗した場合には、代替節がないことが記録されているため、即座に失敗とすることができる。

2.1.2 静的述語宣言を利用した選択点の生成の抑制

インタプリタでは、実行時に assertz などによって追加された新たな節も、後戻り時の代替節として利用される。この機能を実現するためには、述語の最後の節を実行する場合にも選択点を残しておかなければならず、メモリ効率や処理速度が低下するという問題がある。この問題を解決するために、述語を動特性によって静的、動的の2種類に分類し、その動特性を宣言し最適化に利用する方式^[2]を導入している。静的述語は、実行時に定義が変更されない述語で、動的述語は、実行時に定義が変更される可能性がある述語である。一般のプログラムでは、ほとんどの述語が静的述語であり、宣言が行われていない述語は静的述語として扱っているため、宣言が煩雑になることはない。静的述語は、節の追加が行われないことが保証されているので、節を選択した時点で代替節の有無を決定することができる。このため、コンパイルコードと同様に選択点の生成を抑制し、処理速度とメモリ効率を向上させることができる。

2.2 変数分類情報の利用

変数の単一化処理を効率よく行うために、データベース内部における変数項は、その変数の特性を表す以下に示す情報の組合せで表現されている。

- 1) 変数の識別情報(実行時の変数のアドレスに一致)
- 2) 変数の種類(テンポラリ, ローカル, ボイド)
- 3) 変数の出現順位(最初と最初以外)
- 4) 変数のグローバル化の必要性の有無

本インタプリタは、これらの情報を利用してコンパイルコード^[1]と同様な効率のよい単一化処理を行っている。

2.3 環境の生成の抑制

従来の処理系では、複数のサブゴールを持つ節を実行する場合には、環境と呼ばれるデータ構造が必ず制御スタック上に作成される。しかしながら、サブゴールが組み込み述語からなり、いくつかの条件を満足する場合には、環境を作成する必要がない。本インタプリタでは、この条件が満たされる場合には、環境の生成を抑制し処理速度を向上させる方式^[2]を導入している。たとえば、partition/4 の第1節が選択された場合には、最後以外のサブゴールが組み込み述語で、条件も満足されているので環境を生成していない。

2.4 構造複写の抑制

構造複写法では、複合項を利用する場合には、その構造

をデータベースからいったんヒープスタック上に複写しなければならず、複写の処理時間が必要な上に、同様な構造を新たな領域に繰り返し複写しメモリを消費するという問題がある。この問題を解決するために、述語の引数の項を利用される有効範囲によって分類し、その分類情報を用いて構造の複写を抑制する方式^[2]を導入している。

述語の引数として使用される項は、その述語が呼び出された時点で1回だけ使用されるローカル項と、その述語が成功した後も使用され続けるグローバル項に分類することができる。たとえば、is/2 の第2引数は、算術式として is/2 の処理のみで使用されるのでローカル項に分類される。一方、append/3 の第2引数は、第3引数の変数に代入されるリストの一部として、以後の処理で使用されるのでグローバル項に分類される。このような項の分類には、述語が引数をどのように利用するかという情報が必要であるために、組み込み述語のみを対象に行っている。

引数がローカル項に分類される組み込み述語では、引数の複写を行わず、データベース内の原始項をそのまま引数として利用する。したがって、ローカル項を持つ組み込み述語の処理ルーチンは、データベース内部の複合項が引数として直接与えられても処理できるように構成している。

一般的に、構造複写法はメモリ効率が低いとされているが、構造の複写を抑制する本方式は、メモリを使用せず複写時間も必要としないために、構造共有法よりも効果的な実現方式であると考えられる。

3 評価と考察

3.1 処理速度

試作した最適化 PROLOG インタプリタは、C 言語で記述されており、PC-9801VX21 上で稼働している。処理速度の測定は、代表的な3種類のベンチマークプログラムを使用して行った。nreverse, qsort の測定では、repeat, fail ループで 100 回実行させ、その実行時間を 100 で割って処理時間を得ている。queen8 の測定では、すべての解を求めさせて処理時間を得ている。また、他の処理系との比較のために SUN-3/50 上の C-Prolog の処理速度を測定した。表1に示されるように、最適化 PROLOG インタプリタはパーソナルコンピュータ上で稼働しているが、ワークステーション上で稼働している C-Prolog インタプリタと同等以上の処理速度が得られている。2種類の処理系で測定条件が一定でないために、直接比較することはできないが、プロセッサの性能比を考慮すると、最適化 PROLOG インタプリタは、従来のインタプリタの4倍程度の処理速度を達成しているものと考えられる。

表1 ベンチマークテストの処理速度

述語	本インタプリタ		C-Prolog	
	時間(ms)	性能(LIPS)	時間(ms)	性能(LIPS)
nreverse	220	2.2 K	260	1.9 K
qsort	350	1.7 K	360	1.7 K
queen8	43,000	----	53,000	----

3.2 述語の動特性宣言とインデッキング

表2に示されるように、静的述語宣言が行われている場合には、3種類のすべてのプログラムで処理速度とメモリ効率が向上し、最適化の効果が示されている。処理速度は、最大の nreverse で 30% 程度、最小の qsort でも 5% 程度向上している。また、制御スタックのメモリ使用量は、

最高の qsort で 270 分の1以下、最低の queen8 で5分の1になっており、メモリ効率が大幅に向上している。

qsort に対して nreverse ほど宣言の効果が無いのは、qsort から呼ばれている partition/4 の第1節にカットがあり、この節が選択された場合には動的述語であっても選択点が削除され、静的述語として最適化される効果が相対的に低くなるためと考えられる。また、queen8 で最適化の効果がほとんどないのは、生成検査法による本質的に非決定性プログラムとなっているために、静的述語とした場合でも選択点が生成されることが多いためと考えられる。

表2 静的述語と動的述語

述語	静的述語			動的述語		
	処理時間 (ms)	メモリ使用量(Byte)		処理時間 (ms)	メモリ使用量(Byte)	
		制御	トレイル		制御	トレイル
nreverse	220	0.7 K	0	310	20.4 K	1.8 K
qsort	350	0.0 K	0	380	13.5 K	1.2 K
queen8	43,000	0.9 K	0.1 K	45,000	4.3 K	0.2 K

3.3 構造複写の抑制

数値演算を行うだけで、リストなどのデータ構造を作成しないアッカーマン述語を使用して処理時間とメモリ使用量を測定した。従来の方式では、数値演算が行われるたびに算術式の構造が複写されるために、表3に示されるようにヒープスタック領域が多量に消費されているが、構造複写を抑制した方式では、ヒープスタック領域はまったく使用されておらずメモリ効率が大幅に向上している。また、複写処理がないために処理速度も 12% ほど向上している。

表3 構造複写抑制の効果

述語	構造の複写有		構造の複写無	
	処理時間 (ms)	ヒープ使用量 (Byte)	処理時間 (ms)	ヒープ使用量 (Byte)
ack	3,300	43.6 K	2,900	0

4 結論と今後の課題

最適化 PROLOG インタプリタの評価を行い、従来のインタプリタと比較して処理速度が4倍程度になり、メモリ効率も大幅に改善されることを示した。このインタプリタは、32 ビットのプロセッサ上では、1 MIPS あたり 5K LIPS 程度の処理速度が得られ、10 MIPS の処理速度を持つ最近のワークステーション上では、インタプリタでも PROLOG マシン PSI 以上の処理速度が得られるものと考えられる。

今後の課題としては、メモリ効率や処理速度をさらに向上させるために、今回利用しなかった最適化手法を導入し評価を行うことが考えられる。たとえば、ハッシュ法を利用したインデッキングや、レジスタ割当の最適化は、節の登録時の負荷が大きくなりインタプリタの会話性が損なわれるという理由で採用しなかったが、実際にどの程度の負荷が生じるかを評価する予定である。

参考文献

- [1] Warren, D.H.D.: An Abstract Prolog Instruction Set, SRI International, Tech. Note 309, (1983).
- [2] 碓崎 他: 最適化 PROLOG インタプリタの構成法, 情報処理学会, 記号処理研究会資料, 48-4, (1988).