4M-6

論理型言語ＰＲＥＳＴＯの コンパイル方式とその評価

串間和彦　土屋秀幸
（ＮＴＴ　電気通信研究所）

1. はじめに

汎用計算機上に拡張Prolog(=PRESTO)の最適化ｺﾝﾊﾟｲﾗ[1]を開発し、ｵﾌﾞｼﾞｪｸﾄｺｰﾄﾞの性能を実測評価した。ｸｨｯｸｿｰﾄを対象とした評価結果では各種最適化の導入により、約40%の処理時間の向上を達成でき、採用した最適化方式の有効性を確認したので報告する。

2. 最適化方式

Warrenらの旧PLMｺｰﾄﾞ[2]をﾍﾞｰｽとして節のｲﾝﾃﾞｷｼﾝｸﾞ、TRO等の最適化を実現するとともに、評価の結果判明した、性能上のﾎﾞﾄﾙﾈｯｸを解消するために以下の最適化方式を導入した。

2.1 ｺﾋﾟｰ処理の展開

PRESTOは構造体のﾕﾆﾌｨｹｰｼｮﾝ方法としてｽﾄﾗｸﾁｬｰ・ｺﾋﾟｰ方式を採用している。従って処理速度の向上のためには構造体ﾃﾞｰﾀの表現方法を工夫し、ｺﾋﾟｰ処理に適した構成とすることが重要である。そのため、PRESTOでは構造体ﾃﾞｰﾀの表現方法として構造体の構成要素(他の構造体を含む)をｱｰｷﾞｭﾒﾝﾄの順にﾒﾓﾘ上の連続したｱﾄﾞﾚｽに配置する方法をとった。この構成によりｺﾋﾟｰ処理は、

①構造体の置かれた領域をまとめて複写し、

②複写した構造体内にある各変数にｺﾋﾟｰ時の値を結合する　だけで完了する。

しかし、上記処理を任意の構造体のｺﾋﾟｰが可能な汎用のｺﾋﾟｰ・ﾙｰﾁﾝで行うと、構造体の大きさや結合する変数の位置を実行時に決定しなければならず効率が悪い。そこでPRESTOでは、ｺﾝﾊﾟｲﾙ時に構造体の内部を解析して必要な情報(構造体の大きさ、結合する変数の位置)を決定し、ｵﾌﾞｼﾞｪｸﾄとして上記①、②の処理を行う専用のPLMｺｰﾄﾞを生成することで高速なｺﾋﾟｰ処理を実現している(図1)。なお、本最適化の対象となるのは、「内部に変数を含む複合項」のみであり、「変数を含まない複合項」についてはｺﾋﾟｰ処理自体を行わない。

2.2 変数参照の効率化

2.2.1 最適化の着眼点

変数の参照時には、その変数のｽﾀｯｸ上の位置から始めて変数のﾎﾟｲﾝﾄ先を次々に追跡し、真の結合先を決定するﾃﾞﾘﾌｧ処理が必要である。ﾃﾞﾘﾌｧ処理の効率化を目的とする最適化としては、入力ﾃﾞｰﾀを示すﾓｰﾄﾞ宣言(+,++)を利用することでﾎﾟｲﾝﾄ先の追跡を効率化する方法[3]が提案、実現されている。

我々の机上評価の結果でも、①間接参照の機構を持たない汎用機ではﾃﾞﾘﾌｧ処理に多くの処理ｽﾃｯﾌﾟを要すること、②ﾃﾞﾘﾌｧ処理の多くは呼び出し述語から定義節へｱｰｷﾞｭﾒﾝﾄを引き渡す処理において発生していることが判明した。

2.2.2 最適化方法

呼び出し述語から定義節に対してｱｰｷﾞｭﾒﾝﾄを引き渡す方法には、

①各ｱｰｷﾞｭﾒﾝﾄ位置に記述された変数の内容をﾃﾞﾘﾌｧして引き渡す方式、

②ﾃﾞﾘﾌｧを行わずにｽﾀｯｸ上の変数位置のみを引き渡す方式

の2通りがある。①の方式には、定義節側が同じﾃﾞｰﾀを何度も使用する場合、参照に要する処理が減るという利点がある反面、呼び出し側と定義節側の両方でﾃﾞﾘﾌｧ処理が必要という欠点がある。例えば、?-a(X,X).という呼び出し述語に対し、a(1,1).という定義節を考えると、呼び出し側でﾃﾞﾘﾌｧを行っても、第2ｱｰｷﾞｭﾒﾝﾄのﾕﾆﾌｨｹｰｼｮﾝ時には再度ﾃﾞﾘﾌｧが必要である。

我々は変数のﾎﾟｲﾝﾄ先を追跡する処理はたかだか1回必要である場合が多いことに着目し、②の方式の改良案として変数の1回目の出現位置を利用して最適な追跡回数を選択する以下の方式を採用した(図2)。

①呼び出し述語のｱｰｷﾞｭﾒﾝﾄに記述された変数がその変数の節内での2回目以降の出現で、かつ1回目の出現が節頭部である場合には、その変数は常に他の項(変数か定数かは不明)を指していることが明らかなため、無条件にﾎﾟｲﾝﾄ先のｱﾄﾞﾚｽを引き渡す。

②上記以外については結合関係が不明なため、ﾃﾞﾘﾌｧを行わず単に変数位置を引き渡す。

3. 性能評価

3.1 測定結果

Prologｺﾝﾃｽﾄ[4]の例題から6本のTPを選択し、各種最適化の実施前後のCPU時間、ｵﾌﾞｼﾞｪｸﾄ規模を測定した(図3)。またTP2(=ｸｨｯｸｿｰﾄ)に対する所要DSの内訳を机上評価で求めた結果を図4に示す。なお、変数の1回目の出現位置に着目した最適化については標準の処理機構として導入したため、その効果は図4にのみ現れている。

3.2 考察

図3,4から以下がわかる。

①TP2(=ｸｨｯｸｿｰﾄ)の場合、ｺﾋﾟｰ処理の展開によって処理時間は約20%向上している。図4から、主に節頭部のﾕﾆﾌｨｹｰｼｮﾝ処理が効率化されていることがわかる。反面、ｵﾌﾞｼﾞｪｸﾄ規模は約10%増加していることから、ｺﾋﾟｰ処理を展開するか否かは利用者の判断に委ねることが適当と思われる。

②ﾓｰﾄﾞ宣言を利用した最適化は、処理時間、ｵﾌﾞｼﾞｪｸﾄ規模の両方の削減に有効であり、それぞれ約10%、約30%削減されている。机上評価では変数の

Optimization techniques for Prolog Compiler and their effects

Kazuhiko KUSHIMA, Hideyuki TSUCHIYA

NTT Electrical Communications Laboratories

1回目の出現位置に着目した最適化とﾓｰﾄﾞ宣言を利用した最適化による処理時間の削減効果は、(1)両方を同時に行うと約15%、(2)単独で行うと、ともに約10%であった。従って、ﾃﾞﾘﾌｧ処理に適した間接参照の機構を持たない汎用機上ではこれらの最適化はともに有効である。

③TP3、TP4については各種最適化の効果がTP1、TP2ほどあがっていない。これは、両ﾌﾟﾛｸﾞﾗﾑの特性として、ｺﾋﾟｰ処理の展開で対象となる構造体の数が少ない、ﾓｰﾄﾞ宣言の指定が少ないとの理由によるものと思われる。

④TP5、TP6はﾃﾞｰﾀﾍﾞｰｽ検索という問題の性質上ｲﾝﾃﾞｷｼﾝｸﾞの適用により、5～10倍もの大幅な処理時間の短縮が図られている。但し、ｵﾌﾞｼﾞｪｸﾄ規模の点では1.6倍の規模増となっている。

4.おわりに

PRESTOｺﾝﾊﾟｲﾗでは、ｺﾋﾟｰ処理に関する最適化、変数参照に関する最適化等を導入することによりｵﾌﾞｼﾞｪｸﾄ・ｺｰﾄﾞの処理速度の大幅な向上を達成できた。今後ｺﾝﾃｽﾄの例題だけでなく、より大規模なﾌﾟﾛｸﾞﾗﾑを用いた評価を行いたい。

（参考文献）

[1]串間他:"論理型言語PRESTOのｺﾝﾊﾟｲﾙ方式",情処全大,31

[2]Warren,D.H.D.:"IMPLEMENTING PROLOG-compiling predicate logic programs",D.A.I.Research Report,No.39(1977)

[3]岸本他:"Prologｺﾝﾊﾟｲﾗの設計と評価",Proc. of the Logic Prog. Conf. 1985,PP247-258

[4]奥野:"第3回LISPｺﾝﾃｽﾄと第1回PROLOGｺﾝﾃｽﾄの課題案",情処記号処理28-4

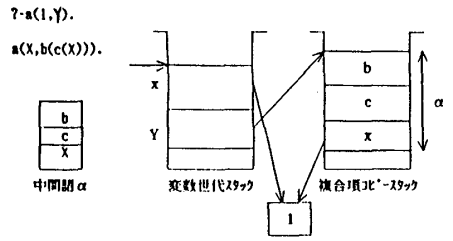
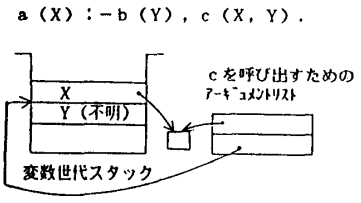


図1 コピー処理の展開



説明：cを呼び出すためのｱｰｷｭﾒﾝﾄﾘｽﾄの作成において、第1ｱｰｷｭﾒﾝﾄには変数Xのﾎﾟｲﾝﾄ先のｱﾄﾞﾚｽを、第2ｱｰｷｭﾒﾝﾄには変数Yのｱﾄﾞﾚｽを格納する。

図2 変数の1回目の出現位置に着目した最適化

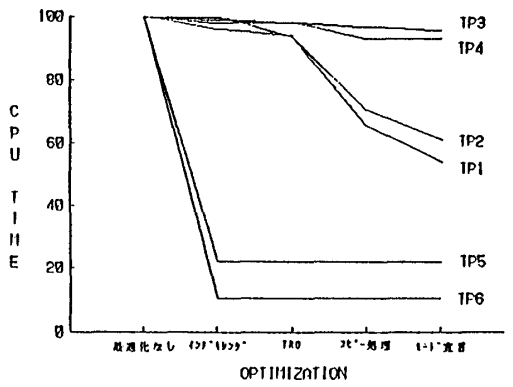


図3－1 各種最適化の実施前後のＣＰＵ時間の評価

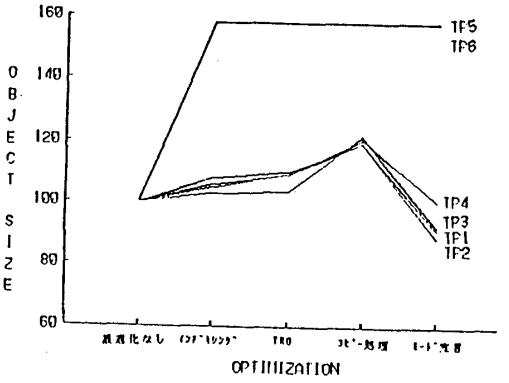
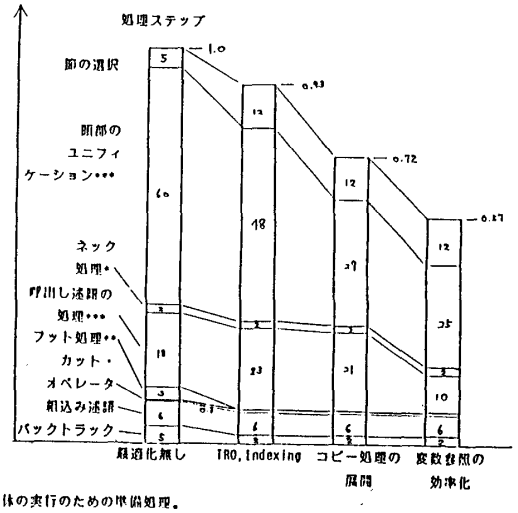


図3－2 各種最適化の実施前後のｵﾌﾞｼﾞｪｸﾄ規模の評価



* 節本体の実行のための準備処理。
** 節からのリターン処理。
*** 変数のデリファはこれらの処理の中で行われる。

図4 クィックソートに対する所要ＤＳの内訳