1Q-5

JSI AIワークステーション(5) —Prologコンパイラの概要と設計方針

小松 秀昭，田村 直之，浅川 康夫，黒川 利明
日本アイ・ビー・エム（株）サイエンス・インスティチュート

1．はじめに

近年 Warren によって考案された抽象マシンに基づいたコンパイラの研究がさかんである。それらは節のインデキシングにおける工夫[2]、レジスタ割り当ての最適化[2][3]、冗長な代入の除去[1][4]、組み込み述語の扱い方[2][4]、変数のクラス分け[1][2][4]、モード宣言の拡張[2][5]、モードや決定性の推論[5]といったものに分類できる。しかし、これらの多くは抽象マシン命令のレベルでの最適化であり、最終的に実行される計算機を考えた最適化ではない。

そこで、我々は汎用計算機をターゲットマシンとして考え、コンパイラの各フェーズにおいて、汎用計算機に有効な各種の最適化を行い、高速実行が可能なコードを生成するPrologコンパイラを設計、試作した。今回、このコンパイラの概要を報告する。

2．従来のアプローチとの比較及び設計方針

Warrenの提案した抽象マシン命令[1][6]は、Prologをレジスタマシン上で効率的に実行するために考えられたもので、基本的にはフレームの作り方が効率的であるなど、非常に優れたものである。

しかし、Warrenの抽象マシン命令は、原則的にはレジスタマシンでありながら、いくつか専用のハードウエアを前提としたものである。例えば、タグの検出及び、その処理は専用のハードウエアによって、わずかなオバーヘッドタイムしか必要としない。また、read/writeモードのために専用のレジスタを使用し、実行時にのチェックにまったく余分な時間を必要としない。さらに、トレイルに関する処理も、専用のハードウエアとスタックメモリのインターリーブによって、並列に行うことを可能としている。

これらの専用のハードウエアは、基本的に実行時のチェックを効率的におこなうものである。Warrenの抽象マシン命令を、そのまま汎用計算機で実現しようとすると、これらの、実行時のチェックやトレイルに関する処理にかなりの時間がかかってしまう。また、抽象マシン命令の1つ1つのレベルが汎用計算機の命令に較べて非常に高度過ぎるため、かなりの最適化の機会を失っていると考えられる。

そこで、我々は汎用機上でコンパイラを実現するにあたって、基本的にはWarrenのモデルを採用しながらも、以下のような変更を加えた処理系を作成した。

- 実行時のチェックのオーバーヘッドを少なくするための最適化
- 十分な最適化が行うことが可能なプリミティブな中間言語
- より多くの最適化を行うためのデータタイプ及び no-trail宣言の導入
- 複数の汎用計算機に対してコード生成が可能なこと

PL.8[7]というシステム記述用の最適化コンパイラをバックエンドプロセッサに用いることによって、複数のターゲットマシンに対してコード生成が可能になる。さらに、PL.8は本来RISCアーキテクチャ用に開発されたコンパイラであるため、非常に高度な最適化能力をもっている。

これまで研究されてきたPrologコンパイラにおける最適化は、そのすべてがProlog に依存したものではなく、レジスタ割り当てや冗長な代入の除去などは、このような既存の最適化コンパイラを用いることで、解決できる。

3．本コンパイラの概要

(1) 全体の処理とデータの流れ

本コンパイラにおける、処理とデータの流れは図1のようになっている。

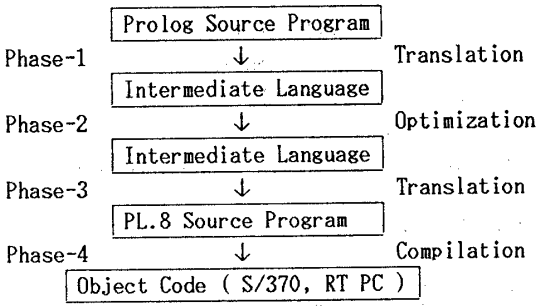

図1：コンパイラの概略

フェーズ1では、 Prolog のソースプログラムを WIL とよばれる中間言語に変換する。フェーズ2では、WILプログラムの最適化を行う。フェーズ3では、WILプログラムからPL.8のプログラムをターゲットマシンごとに生成する。

JSI AI Workstation(5) - An abstract of the Prolog Compiler
Hideaki Komatsu, Naoyuki Tamura, Yasuo Asakawa, Toshiaki Kurokawa
Science Institute, IBM Japan, Ltd.

(2) 中間言語の説明

WIL は Warren の抽象マシン命令をベースにした中間言語である。本コンパイラでは、汎用計算機をターゲットマシンとするために、以下のような拡張を行っている。

・高度な最適化が行えるように、Warren の抽象マシン命令をより細かな命令に分解している。これによって、従来行われてきた最適化の機会を増やしている。

・最適化のために必要なデータフロー解析を、有効に行うためのタイプなどの付加的情報を付け加えている。

(3) 各宣言の導入

DEC-10 Prolog[8]で導入されたモード宣言は、Prologの述語の使われ方をコンパイラに知らせる方法として有名であるが、我々はあらたにタイプ宣言とnotrail宣言の２つの宣言を導入した。

データタイプは、実際にはusage宣言の中でモードと合わせて宣言される。述語ごとに補足的に付けられるusage宣言では、その述語の各引数がどういうタイプのデータを、入力または出力として扱うかを宣言できる。宣言できるデータタイプは、atom integer list nil structure variableといった基本的なデータタイプと、それらの任意の組み合わせに制限している。空リスト(nil)のはlistと異なったデータタイプとしたほうが、リスト処理の最適化にとって有利であると考えて、独立したデータタイプとした。

notrail宣言は決定的にしか使われない述語に対して、トレイルの処理すら必要ないことを宣言するものである。通常、usage宣言と合わせて、関数的な述語に付けられる。例えば、２つのリスト（空リストの可能性もある）を入力として受け取り、それらを結合したリストを返す述語appendは図３のように定義できる。

```
<- usage append( in:(list+nil),
                 in:(list+nil),
                out:(list+nil)).
<- notrail append(*, *, *).
append({}, L, L).
append({X|L1}, L2, {X|L3}) <- append(L1, L2, L3).
```

図３：宣言付きappend

４.本コンパイラにおける最適化の概要

本コンパイラの各フェーズにおいて行われている最適化は次のようになっている。

(1) フェーズ１での最適化

フェーズ１では、従来のPrologコンパイラで行われてきた、Prolog言語に依存した最適化をおこなう。

・入力モードの引き数から優先的に統一化をおこなう。
・タイプ、モードの情報を使い "unsafe" な変数を減らす。
・タイプ情報を使って最適な組み込み述語を選択する。
・処理の軽い組み込み述語を一般の述語呼び出しと区別することにより、変数のクラスを効率的なものにする。

(2) フェーズ２での最適化

フェーズ３では、データフロー解析を行い、ターゲットマシンに依存しない最適化を行う。

・分岐命令の前条件から考えて、選択され得ない分岐先のチェック命令の削除
・分岐命令に入ってくるパスに付随している条件のもとでは、常に決まった所にしか分岐しない場合、その分岐先にパスを付け換える。
・実行され得ない命令を削除する。

(3) フェーズ３での最適化

フェーズ３では、ターゲットマシンの情報を用いて、マシンに依存した最適化をおこなう。

・タグの処理（セット、付け換え）をマシンに合わせて最適化する。
・コスト評価によって、分岐の探索順序を変更する。（マシンに依存したインデキシングの最適化）

４．まとめ

今回試作したPrologコンパイラの概要を述べた、具体的な最適化処理のより具体的な説明、及び本コンパイラの実際の評価はこの後の報告で述べる。

参考文献

[1] Warren, D.H.D., "An Abstract Prolog Instruction Set", SRI International Technical Note 309, October,1983.

[2] 黒沢他、"Prologコンパイラの開発(2)-(4)"、情報処理学会第32回全国大会、pp.377-382、1986

[3] 寺門・玉木、"Prologコンパイラにおける最適化技法"、情報処理学会第32回全国大会、pp.391-392、1986

[4] Park S. and Degroot D., " Clause-Level Optimization of Abstract Prolog INstruction Set", Reserch Report RC11320, IBM Thomas J.Watson Research Center, 1985.

[5] Mellish, C.S., " Some global optimization for Prolog compiler", J. OF LOGIC PROGRAMMING, 1985 No.1 pp.43-66.

[6] Tick,E. and Warren D.H.D., "Towards a Pipelined Prolog Processor", Proc. of 1984 International Symposium on Logic Programming, IEEE Computer Society, 1984.

[7] Auslander, M. and Hopkins, M., "An Overview of the PL.8 Compiler ", Proceedings of SIGPLAN '82 Symposium on Compiler Construction, Volume 17, Number 6, June 1982.

[8] Warren, D.H.D., " Implementing Prolog - compiling predicate logic program ", Reserch Reports 39 & 40, Dept. of Artificial Intelligence. Univ. of Edinburgh, 1977.