3U－4

プログラム変換に基づく計算システムのためのコンパイラと仮想マシンの作成

中尾 公俊　赤間 清　宮本 衛市

北海道大学 工学部

1 はじめに

我々はこれまで複雑な問題解決や知識処理のための計算モデルとして「ルールを基礎としたプログラム変換」RBPT(Rule Based Program Transformation)を提案してきた [1]。

このモデルは、ルールによるプログラム変換を計算とみなすモデルである。このモデルの利点は、計算の正当性を厳密に議論でき、計算の順序を問う必要がないという点である。

この計算モデルに基づく言語として、我々の研究室では、RBPTI(RBPT Interpreter) を開発した。これを用いた様々な実験によりその有用性が確認されている。しかし、RBPTI は Prolog のメタインタプリタとして実現されており、実行速度の面で改善すべき点が残されている。

本論文では、RBPTI の実行速度に関する問題を解決するために作成した、RBPTI のコンパイラと仮想マシンについて報告する。

2 プログラム変換ルール

プログラム変換ルール (以降、ルール) はルール名と、次の 4 つの要素から構成される。変換対象を記述する Head 部、ルールを適用するための条件を記述する Cond 部、ルール適用に際して必要な実行を記述する Exec 部、変換後のアトムを記述する Body 部からなる。

ルールは、変換対象と Head 部が一致し、Cond 部の条件を満たした時適用され、Exec 部を実行し、Head 部を Body 部に置き換え、終了する。

例えば、*num に数を与えると、*num 個の要素からなるリストを作るルールは図 1 のように記述される。

このルールは、変換対象の第 1 引数が正である時に適用される。

このルールと終止条件を扱うルールのもとで (gen 5 *list) を初期プログラムとしてプログラム変換を行なうと、5 回、ルールが適用され、結果として、(gen 5 (5 4 3 2 1)) を得る。

A Compiler and an Abstract Machine for Rule Based Program Transformation
Kimitoshi NAKAO, Kiyoshi AKAMA, Eiichi MIYAMOTO
Hokkaido University
Nishi8,Kita13,Kitaku,Sapporo,Hokkaido 060,Japan

```
(Rule gen1
    (Head (gen *num *list))
    (Cond (> *num 0))
    (Exec (:= *NUM (- *num 1))
          (unify *list (*num . *L)))
    (Body (gen *NUM *L)))
```

図 1: ルールの記述例

3 RBPTI の問題点と解決

3.1 RBPTI の実行速度の問題

RBPTI の実行速度が遅くなる要因は、

(1) メタインタプリタによる実行

(2) 同一の Head 部を持つルールにおける Cond 部の重複した条件判定や、不必要な条件判定

である。

(2) について説明する。Head 部が同一である 3 つのルール、rule1 、rule2 、rule3 があると仮定する。これらのルールの Cond 部は Cond A 、Cond B が共通で、さらに rule1 には Cond C 、rule2 には Cond D 、rule3 には Cond E が含まれているとする。また、Cond D と Cond E は互いに排反とする。

RBPTI における上述のルール選択の過程を図 2 に示す。

上述のルールと Head 部が一致し、Cond A 、Cond B 、Cond E を満たす変換対象があると仮定する。RBPTI では、この条件判定を以下の手順で行う。rule1 の Cond A 、Cond B の条件判定は成功するが Cond C で失敗し、rule1 は適用できない。rule2 の Cond A 、Cond B の条件判定は成功するが Cond D で失敗し、rule2 は適用できない。rule3 の Cond A 、Cond B 、Cond E の条件判定は成功し、9 回の条件判定の後、rule3 が適用可能であるとわかる。

この方法の問題点は以下の 2 点である。(a) 各ルールにおいて Cond A と Cond B を重複して条件判定している。(b)Cond D と Cond E は互いに排反なので、Cond D の条件判定に失敗した時点で Cond E

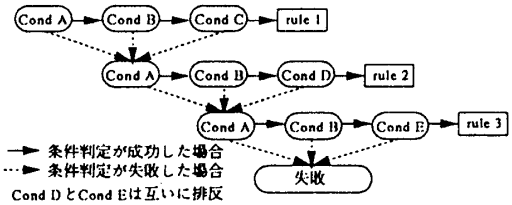

図 2: RBPTI の条件判定の過程

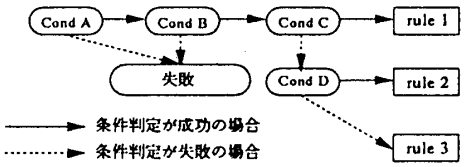

図 3: 決定木を用いた条件判定の過程

が成功することがわかるが、rule 3 において、Cond E の条件判定を再び行なっている。

3.2 問題点の解決

(1) はコンパイラを用いることで解決できる。本論文では、このコンパイラを RBPTC(RBPT Compiler) と呼ぶ。(2) は同一の Head 部を持つルールの Cond 部から決定木を生成することにより解決できる。

4 RBPTC と仮想マシン RBPTM

4.1 決定木を用いた高速化

RBPTC は、同一の条件判定の重複と、互いに排反である条件の不必要な判定を行なわないような決定木を作り、これに基づいて仮想コードを生成する。決定木のノードには Cond 部の条件が記述される。図 3 は 3.1 節の rule1 、rule2 、rule3 を決定木で表したものである。条件判定は Cond A から始める。条件が満たされる場合には実線を、満たされない場合は点線をたどり、適用可能なルールを選択する。

4.2 判定回数の比較

rule 3 が選択されるまでに、RBPTI では 9 回の条件判定が必要であったが、決定木を用いることにより 4 回の条件判定に短縮された。

4.3 仮想マシン RBPTM

RBPTC によって出力された仮想コードを動作させる、仮想マシン RBPTM(RBPT Machine) は、Warren の仮想マシン WAM [2] にプログラム変換のための機能を付加して実現される。

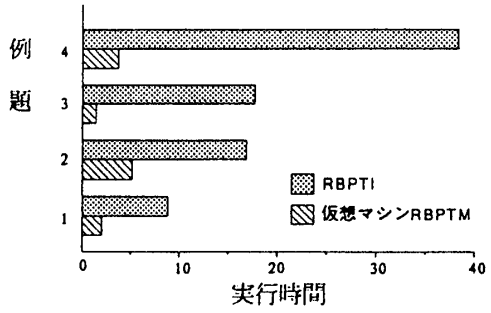

図 4: RBPTI とコンパイラの実行時間の比較

5 評価

RBPTC と仮想マシンの効果を調べるため RBPTI との比較を以下の例題において行った。結果を図 4 に示す。

1. 500 個の要素からなるリスト A を作り、リスト A とリスト A を結合したリストを得る。
2. 1000 個の要素からなるリスト B を作り、リスト B とリスト B を結合したリストを得る。
3. 自然言語の意味解析 1
4. 自然言語の意味解析 2

例題 1 と例題 2 では RBPTI と比べ 3 倍前後の高速化が見られらた。

また、例題 3 と例題 4 では RBPTI と比べ 10 倍前後の高速化が見られらた。

この結果から、RBPTC と仮想マシンの有効性が確認された。

6 終りに

本研究の目標である RBPTI の高速化は、同一の Head 部を持つルールの Cond 部から決定木を生成するコンパイラと、仮想マシンを作成することにより達成することができた。しかし、まだ改良の余地がある。例えば、コンパイルコードの最適化、変換対象の格納方式の再検討、決定木を生成するときの検索の強化などである。今後、更なる高速化のために、以上の点について研究する必要がある。

参考文献

[1] 赤間清:ルールを基礎としたプログラム変換,「自然言語処理における実働」シンポジウム論文集,pp.86-94(1994)

[2] David H.D. Warren:An abstract Prolog instruction set.Technical Note 309.SRI International,Menlo Park,CA,Octorbar(1983)