部分計算によるプロダクションシステムの高速化
Deriving an Efficient Production System by Partial Evaluation

古川康一 Koichi Furukawa
藤田博 Hiroshi Fujita
新谷虎松* Toramatsu Shintani*
(財)新世代コンピュータ技術開発機構 ICOT
富士通* Fujitsu*

A great deal of research has been done on applying partial evaluation for optimizing meta-programs in Prolog, such as a rule interpreter with certainty factor handling capability, a bottom-up parser and a formula manipulation system. It has been claimed that the technique is very useful in developing various inference engines for knowledge based systems. However, nobody has succeeded in deriving an efficient production system (PS) through partial evaluation. This paper presents the derivation of an efficient PS by partially evaluating a simple PS interpreter, given a set of rules. The derived codes are shown to be very similar to the compiled codes given by [Shintani 88].

1 はじめに

インタプリタと部分計算器を組み合わせて,知識ベースシステムの種々の推論エンジンを作る試みが,各所で始められている.それは,ルール・コンパイラを作る従来の方法に取って代わるものと考えられている.新しいアプローチは,コンパイル処理全体を二つの部分に分けている.その一つは,ルールの解釈で,他の一つはその部分計算である.この分割によって,コンパイル処理全体が直接コンパイルをする従来のアプローチに比べて格段に理解し易くなっている.さらに,このアプローチによれば,虫取りや修正などのプログラムの維持管理が大変やり易くなる.

部分計算によってコンパイラを実現するというアイディアは,[Futamura 71]によって提案され,小規模の実験は行われていたが,あまり大きな発展は見られなかった.竹内等[Takeuchi 86]は,論理型言語の枠組でこの考えの実証を試みた.その結果,このアプローチが十分に実用的価値を有することが示された.このことは,論理型言語がこのアプローチに適していることを物語っている.その理由は,論理型言語ではmeta-programmingの手法を用いることによって簡単にインタプリタを書くことができること,および同じmeta-programmingの手法によって簡単に部分計算器を実現できること,の二つである.今までに,この枠組の中で,色々な応用がなされてきた.それらは,確信度つきのルール・インタプリタ,ボトムアップ・パーザ,多世界知識ベースなどである.プロダクションシステムについても,部分計算による効率化がこれまでも試みられてきたが,Prologとの制御構造の大きな違いにより,十分な効率化が達成できなかった.

一方,新谷[Shintani 88]は,Prologの機能をうまく活かしたプロダクションシステムの開発に成功した.それはコンパイラ方式であるが,作業領域(Working Memory, WM)内の要素によってルールが選択的に起動されるようになっている.我々は,新谷のアルゴリズムがインタプリタと部分計算の組み合わせで実現できることを見出した.本論文では,そのインタプリタ,および部分計算の概要について述べる.また,ルービック・キューブを解くためのルール集合を与えてそのインタプリタを部分計算した時の性能測定結果を,あわせて示す.

2 Prologによるプロダクションシステムの記述

Prologによってプロダクションシステムを実現する最も簡単な方法は,WMを述語の引数で表わし,ルールを単位節で表わす方法である.そのような記述の例を図-1に示す.この図で,`ps(WM,FinalState)`は,`WM`を作業領域と

し、FinalStateを最終状態とするプロダクションシステム・エンジンを示す。図から分かるように、psの第二節はrecognize(WM,RHS)およびact(RHS,WM,NewWM)を再帰呼び出しによって繰り返し実行するプログラムである。

```
:- op(150,xfy,=>).

ps(WM,FinalState) :-
        member(FinalState,WM).
ps(WM,FinalState) :-
        member(Fact,WM),
        recognize(WM,RHS),!,
        act(RHS,WM,NewWM),
        ps(NewWM,FinalState).

recognize(WM,RHS) :-
        rule(LHS => RHS),
        deduce(LHS,WM).

deduce([],WM).
deduce([C|Cs],WM) :- deduce1(C,WM),
                     deduce(Cs,WM).

deduce1(call(X),_) :- call(X).
deduce1(X,[X|_]) :- member(X,WM).

member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).

act([],WM,WM).
act([Act|As],WM,New_WM) :-
        act1(Act,WM,Int_WM),
        act(As,Int_WM,New_WM).

act1(replace(X,Y),[],[]).
act1(replace(X,Y),[X|L],[Y|L]).
act1(replace(X,Y),[Z|L],[Z|L1]) :-
        act1(replace(X,Y),L,L1).
act1(call(X),WM,WM) :- call(X).
```

図-1 単純PSインタプリタ

recognize(WM,RHS)は、現在の作業領域の状態WMで起動されるルール"LHS=>RHS"、すなわちdeduce(LHS,WM)が成り立つルールを探しだし、その右辺RHSを答として返す。act(RHS,WM,NewWM)は、WMに対して更新コマンド列RHSを作用して、新たな作業領域の状態NewWMを得る。psの再帰呼び出しは、その第一引数をNewWMとして行う。psの第一節は、ループの停止条件を表わす。すなわち、WM中に、あらかじめ与えておいた最終状態FinalStateが現れた時、このpsは停止する。

deduceおよびactの詳細な説明は省略するが、その両者とも、call述語によって、Prologの任意のプログラムを呼び出すことができる。

今、recognizeプログラムを対象としてルールを与えて部分計算することを考える。ところが、このプログラムは、部分計算を行っても効率の向上を図ることができない。それは、WMが与えられないと適用可能なルールが選び出せないので、部分計算時に適用可能なルールを絞り込めないからである。

ところで、WMの値を仮定すると、部分計算時のルールの絞り込みが可能になる。新谷のアルゴリズムは、ルールを絞り込むのにWMの一つの要素だけを用いる。すなわち、recognize述語にWMの要素を表わす引数を追加して、それを手掛りにしてルールの選択を行うようにする。

このような変更を行って得られたプロダクションシステムを、図-2に示す。図-1のプロダクションシステムを単純プロダクションシステムと呼び、図-2のそれをWM駆動プロダクションシステムと呼ぶことにする。図-2のプログラムのrecoginizc(Fact,WM,RHS)の本体部は、rule, del, およびdeduceの三つのgoalからできている。すなわち、rule(LHS=>RHS)の呼び出しによって、ルール・ベースからルールが順次取り出され、del(Fact,LHS,NewLHS)によって、与えられたWMの要素FactがLHSに含まれているかどうかを調べ、もし含まれていたら、その要素をLHSから取り除いて、残りのリストNewLHSを求める。そして、その残りのリストに対して、deduceを実行する。

さて、このように変更されたWM駆動プロダクションシステムを解釈実行しても、実は何の効率化も達成されない。それは、依然として適用可能なルールが見つかるまでルール・ベースを順次探さなければならないからである。しかしながら、このプロダクションシステムは、部分計算による効率化が可能である。それを、次節で示そう。

3 WM駆動プロダクションシステムの部分計算

WM駆動プロダクションシステムのrecognize述語を、ルール

```
rule([ goal(put1),t(1,1)]
     => [replace(goal(put1),
         goal(put2))]).
```

に特化することを考える．この部分計算の様子は，recognize(Fact,WM,RHS) の記号計算から観察できる．それは次の通りである．

```
?-recognize(Fact,WM,RHS).
   |
   V
?-rule(LHS => RHS),
  del(Fact,LHS,NewLHS),
  deduce(NewLHS,WM).
   |
   |  <LHS = [goal(put1),t(1,1)],
   |   RHS = [replace...]>
   V
?-del(Fact,[goal(put1),t(1,1)],NewLHS),
  deduce(NewLHS,WM).
   |
   |  <Fact = goal(put1),
   |   NewLHS = [t(1,1)]>
   V
?-deduce([t(1,1)],WM).
   |
   V
?-deduce1(t(1,1),WM).
```

記号計算で最後に残ったゴールが部分計算時に実行できなかったゴールなので，部分計算の結果は，次のようなプログラムとなる．

```
recognize(goal(put1),WM,
  [repl(goal(put1),goal(put2))]) :-
   deduce1(t(1,1)).
```

del 述語は非決定的であるので，部分計算の結果，上の節の他に次のような節も同時に得られる．

```
recognize(t(1,1),WM,
  [repl(goal(put1),goal(put2))]) :-
   deduce1(goal(put1)).
```

さて，このようにして得られた recognize プログラムは，その第一引数を LHS 部に含むようなルールを展開したものに限られている．すなわち，もし実行時に第一引数に具体値が与えられると，recognize はそれを LHS 部に含むルールしかアクセスしないことになる．これは，部分計算によって，recognize がアクセスすべきルールの集合を WM の要素によって絞り込んだことを意味する．

ここで，部分計算の実行時に変数 Fact に具体値を与えなかった点に注意しよう．部分計算は，通常既知の情報を引数を経由して指定するが，Prolog では，データベースの情報が論理変数に代入されることによって，既知情報を部分計算の実行中に引数に与えることができる．これは，論理変数の特徴であり，具体化における値の後方伝播として知られている．

```
ps(WM,FinalWM) :-
        member(Fact,WM),
        recognize(Fact,WM,RHS),
        act(RHS,WM,NewWM),
        ps(NewWM,FinalWM).
ps(FinalWM,FinalWM).

recognize(Fact,WM,RHS) :-
       rule(LHS=>RHS),
       del(Fact,LHS,NewLHS),
       deduce(NewLHS,WM).

del(X,[X|Y],Y).
del(X,[A|Y],[A|Z]) :- del(X,Y,Z).
```

図-2 WM 駆動型 PS インタプリタ

4 性能評価

部分計算による効率の向上がどの程度達成されるかを見るために，我々は，ルービック・キューブを解くためのルール集合を与えて，プロダクションシステムの特殊化を行った．問題の大きさを変えるために，問題全体を 5 段階に分けた．それらは，一面の辺，完全一面，完全二層，完全頂点，完全キューブ，の五つである．我々は，recognize 節のみならず，act 節についても部分計算を行った．それは，主に act で呼び出されるキューブの回転操作のための Prolog プログラムの計算である．表-1 に，ルービック・キューブを解くために要した時間を測定した結果を示す．ここで，第一行は，各段階まで解くのに必要とするルールの数を示す．第二行 (1) は，単純プロダクションシステムでの計算時間である．第三行 (2) は，WM 駆動プロダクションシステムで，act 中の call 述語を部分計算して得られたプログラムによる計算時間である．第四行 (3) は，さらにそれを recognize について部分計算した結果のプログラムによる計算時間である．図-3 に，(1)/(2) および (2)/(3) のグラフを示す．これらは，それぞれ act および recognize の最適化によってもたらされた効率改善比を表わす．このグラフから分かるように，特に recognize の部分計算は，ルール数に比例した効率の向上をもたらす．これは，単純プロダクションシステムでは recognize の処理がルール数に比例して増大するのに対して，それを部分計算すると，各時点でアクセスされるルール (実際にはルールが埋め込まれた recognize 節) の数がほぼ一定になると予想されるからである．

表-1 ルービックキューブ実行例

段階	一面の辺	完全一面	完全二層	完全頂点	完全キューブ
ルール数	15	27	33	40	61
(1) 単純PS*	400	600	1119	1520	2659
(2) act のみ特化*	280	520	879	1219	1879
(3) recognize+act*	219	299	400	479	560

*CPU時間(msec) (SICSTUS-Prolog on SUN-3)

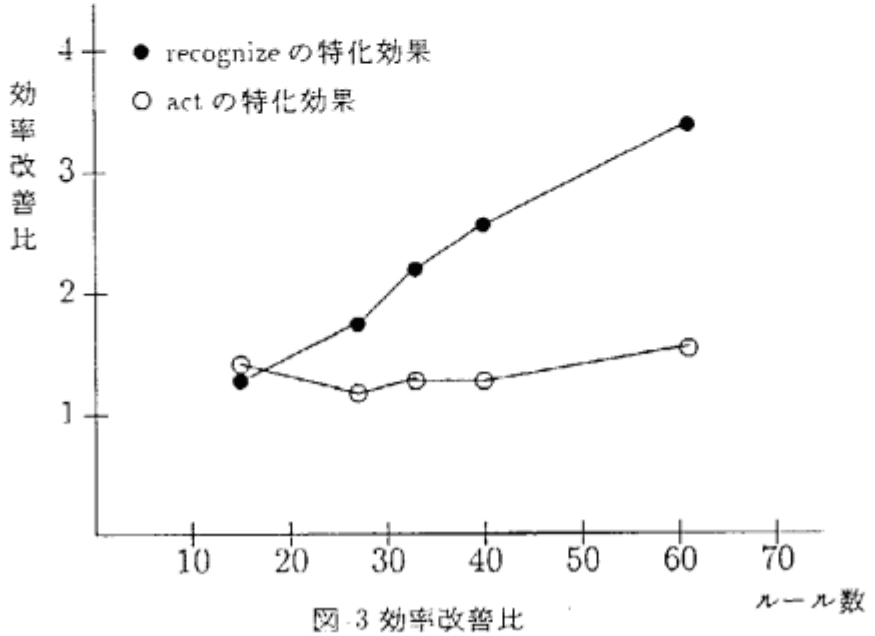

図-3 効率改善比

5 おわりに

本論文では，プロダクションシステム・インタプリタを与えられたルール集合に対して部分計算により特殊化することによって，処理を高速化する方法を与えた．そして，その効率は，ルール数に比例して向上することが示された．これは，単純な部分計算ではアルゴリズムのオーダを変えるような効率の向上は達成できない，というこれまでの常識を覆すものである．

本論文では述べなかったが，競合解消を行うインタプリタの開発も行った．また，モンキーバナナの例で，効率の比較を行ったが，その改善は，30%に止まった．この場合も，ルール数の増加によって，改善率はもっと向上すると考えられる．

今後の課題としては，このアプローチの延長として，プロダクションシステムの並列化を検討している．部分計算の結果得られたプログラムは，ルールを解釈するメタ的な部分がなく，かつassert, retractなどの副作用を伴う処理を含んでいないので，並列化に適していると考えられる．

参考文献

[Futamura 71] Futamura, Y., "Partial Evaluation of Computation Process - An Approach to a Compiler-Compiler," *Systems, Computers, Controls* 2(5) pp.45-50, 1971.

[Shintani 88] Shintani, T., "A Fast Prolog-based Production System KORE/IE," Kowalski, R.A. and Bowen, K.A. (eds.), *Proc. of the Fifth International Conference and Symposium on Logic Programming*, MIT Press, pp.26-41, 1988.

[Takeuchi 86] Takeuchi, A. and Furukawa, K., "Partial Evaluation of Prolog Programs and Its Application to Meta Programming," Kuger H.J. (ed.), *Information Processing 86*, Dublin, Ireland, North-Holland, pp.415-420, 1986.