ICOT Technical Memorandum: TM-0135

TM-0135

数式処理システム AMIE への部分計算の応用とその評価

武脇敏晃，竹内彰一
國藤　進，古川康一

September, 1985

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

数式処理システムAMIEへの部分計算の応用とその評価

Application of Partial Evaluation to the Algebraic Manipulation System AMIE and its Eavaluation

武脇敏晃　竹内彰一　國藤進　古川康一
T. Takewaki　A. Takeuchi　S. Kunifuji　K. Furukawa

(財)新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

数式処理システムAMIEは、メタプログラミング技法を用いてシステムを構築している。メタプログラミングを用いると、プログラムが読みやすい、修正しやすいという利点がある反面、実行速度が遅いという欠点がある。部分計算は、この欠点を取り除くことができる有力なツールである。ここでは、AMIEに部分計算を応用し、特殊化されたプログラムの評価を行なった。

1. はじめに

メタプログラミング技法を用いたものとしてShapiroのAlgorithmic Program Debugging [shapiro 83a]などがあり、その有用性が確認されつつある。しかし、メタプログラミング技法の欠点として、処理速度が遅いという点がある。この欠点を補うのもとして、竹内らによる部分計算(PEVAL) [Takeuchi 85a, 85b] がある。図1に示されるように、PEVAL はメタインタプリタにオブジェクトプログラムを与えた形で部分計算を行い、特殊化されたプログラムを生成する。

本論文では、この部分計算を数式処理システムAMIE [Takewaki 84, 85] に応用し評価したので以下に報告する。

まず本論文では、2,3,4章においてメタプログラミング技法を用いて開発した数式処理システムの制御について述べる。そして、5章においてcut やnot を含んだプログラムの部分計算について述べる。最後に、数式処理システムの部分計算の効率化について述べる。

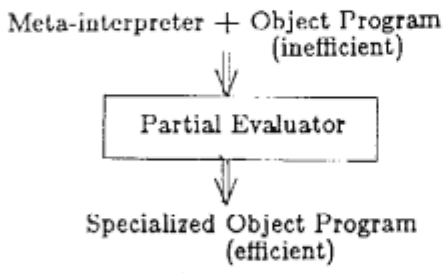

図1メタプログラミングの部分計算

2. 数式処理システムAMIE

AMIEは、メタプログラミング技法を用いて開発したが、その基本となったのはdemo述語[Bowen 81, Furukawa 84] である。このシステムにおいては、述語solve がdemo述語に相当する。

AMIEのオペレータは、メソッドと呼び、各々のメソッドは、それに属する書換え規則集合と、それを数式に適用するためのプログラムから成っている。数式は、このメソッドを適用することによって解かれていく。AMIEは、分離メソッド、収集メソッドなど幾つかのメソッドを持っており、基本的なメソッドの多くはPRESS [Bundy 81]のものと類似している。

3. メタ推論とメタ知識

メソッドを効率良く選択し、入力式に書換え規則を適用して行くために、メタ推論の技法を用いている。このシステムで用いたメタ推論は、メタな知識を用いてどのメソッドを選択すれば良いかを推論する。このメタ知識には、次のような数式に関するメタ概念を用いている。

☆　未知数の数
☆　未知数の存在する位置
☆　数式に含まれる関数の種類
☆　多項式の次数
☆　数式に含まれる演算子　　など

```
strategy(LHS=RHS,isolation,Unknown,Control,
                    [position(P)|Control]) :-
        free_of(Unknown,RHS),
        singleocc(Unknown,LHS),
        position(Unknown,LHS,P).
strategy(LHS=RHS,factorization, Unknown,
                              Control,Control) :-
        main_op_mult(LHS),
        zero(RHS).
                :
```

図２　述語strategyの一部

これらのメタ概念を組合せることによって、各々のメソッドの選択を行う。

このメソッドの選択を行う述語strategyの一部を図2に示す。

第１節は、方程式の右辺RHS に未知数Unknown を含まず(述語free-of)、方程式の左辺LHS に未知数を１つだけ含み(述語singleocc)、左辺に含まれる未知数の位置がわかる（述語position) 時に、分離(isolation) メソッドを選択する。

第２節は、左辺の主演算子が乗算であり(述語main-op-mult) 、右辺が０(述語zero) の時に、因数分解(factorization) メソッドを選択する。

この述語strategyは、数式全体の流れを制御するmethod-demo において用いられる。図 3に、述語method-demo を示す。（ただし、説明機能を抜いた簡略版）

述語method-demo のそれぞれの引数は、解くべき入力式、未知数、解の有効範囲などを示す制御情報、これまでの解法履歴、そして、入力式の解を示している。以下に、各節の説明を行う。

MD1)は、これまでの履歴History の中に入力式と同じ数式が存在しないか調べる。もしその様な数式が存在したならば、無限ループに陥る可能性があるため、'Find infinite loop'のメッセージを返して、以後の処理を中断する。

MD2)は、数式がor演算子で結合されている時に、それぞれの数式Exp1,Exp2 を分割して処理を行うものである。or演算子で結合された数式は、因数分解メソッドを適用した時に生成される。

MD3)は、入力式が解けたか否かを調べるものである。例えば、入力式が方程式の場合は、X=A (Aには未知数を含まず、X は未知数) の時、また、微分，積分の場合には、それを示す関数(def,int) が入力式に含まれなくなった時である。この時に、Control に含まれる制約条件によって正しい解であるかを判定する。例えば、 $X^2 - 4 = 0$ を解くと、解の候補として$x = \pm 2$が得られるが、X > 0 なる制約条件がある時にはX = -2を正しい解とはしない。

MD4)は、述語strategyによって入力式に適用するメソッドを選択し、述語solve によって書換え規則の適用を行う。そして、述語method-demo によって、再帰的に数式の処理を行う。

MD4 における述語solve は、cut を含むプログラムを扱うメタインタプリタであり、このシステムにおいては、メソッド名とdomain world名は１対１の対応を持っている。つまり、domain world名を切り替えることによって、入力式に対して適用する書換え規則の集合を切り替えている。

4. メタプログラミング

メタプログラミング技法は、強力なプログラミング環

```
method_demo(Expression,_,_,History,'Find infinite loop') :-
          loop_check(Expression,History).
method_demo(Exp1 or Exp2, Unknown,Control,History,Ans1 or Ans2) :-
          method_demo(Exp1,Unknown,Control,History,Ans1),
          method_demo(Exp2,Unknown,Control,History,Ans2).
method_demo(Expression,Unknown,Control,History,Answer) :-
          end_check(Expression,Unknown,Control,Ans).
method_demo(Expression,Unknown,Control,History,Answer) :-
          strategy(Expression,World,Unknown,Control,NewControl),
          solve(World,
            apply(Expression,Unknown,New_Exp,NewUnknown),
            CutMark),
          method_demo(New_Exp,New_Unknown,NewControl,
              [Expression|History],Answer).
```

図３　述語method-demo

境を構築することが可能である。そして、多くのメタプログラムは、オブジェクトプログラムのインタプリタの動作を変更すること、すなわち、推論の戦略を変更することも容易に行うことができる。このことからも、メタプログラミング技法を行なえば、容易にプロトタイプシステムを開発することが可能となる。このメタプログラミング技法による特徴をまとめると、次のようになる。

利点

(1) メタ推論部とオブジェクト推論部の制御を明確に分離することができる。

(2) このことによって、メタプログラムとオブジェクトプログラムも分離され、プログラムの読み易さも増加する。

(3) また、規則の追加、修正といったプログラムの変更が容易になる。

欠点

(1) オブジェクトプログラムは、メタインタプリタによって逐次解釈実行されるために実行速度が遅くなる。

図 4に示すメタインタプリタは、部分計算しやすいように書き直してある。これについては、説明を省略する。

5. 部分計算の応用

竹内の部分計算器(PEVAL) は、Pure Prolog を対象

```
solve(World,(P;Q),V):-
        solve(World,P,V); solve(World,Q,V)).
solve(World,(!,Q) ,V) :- cut(V),
        (V==cut,! ; solve(World,Q,V)).
solve(World,(P,Q) ,V) :- P \== !,
        solve(World,P,V), solve(World,Q,V).
solve(World,!     ,V) :- cut(V).
solve(World,true  ,_).
solve(World,not(P),_) :-
        if(solve(World,P,_),fail,true).
solve(World,P     ,V) :-
        clause_world(World,P,Q),
        solve(World,Q,V),
        (V==cut,!,fail;true).
solve(World,P     ,V) :- system(P),
        call(P).

cut(_).
cut(cut).

clause_world(World,P,Q):-X=..[World,R],
        X,
        clause_body(R,P,Q).

clause_body((P:-Q),P,Q   ) :- !.
clause_body(P     ,P,true).
```

図4　部分計算に向いたメタインタプリタ

としているためcut,not を含むプログラムをそのままの形で扱うことはできない。つまり、cut,not を含むプログラムをif-then-else形式のプログラムに変換を行う。例えば、次のように行う。

```
not(P)  →  if(P,fail,true)

member(X,[X,  ]) :- !.
member(X,[_,T]) :- member(X,T).
                ⋮
member(X,[H | T]) :-
        if(X=H, true, member(X,T)).
```

しかし、cut が本質的な意味を持つ場合には、if-then-else形式のプログラムに変換することは難しい。そこで、述語定義の中でcut が現れている場合には、その述語のunfolding を禁止しなければいけない。図 5は、(1) のオリジナルプログラムに対して、(2) すべての述語をunfolding して部分計算したものと、(3) cut を含む述語のunfolding を禁止して部分計算したものを示している。(1),(3) のプログラムは、同様の振舞いをする。しかし、(2) のプログラムのcut の意味は、部分計算を行ったことによって変化している。例えば、a の第１節が実行され、b1が成功し、b2が失敗したとする。(1),(3) の場合、cut によってb が失敗となり、a の第２節が実行される。しかし、(2) の場合には、cut によってa 全体が失敗してしまう。このように、cut を含む述語のunfolding を禁止することは、メタインタプリタにおいても同様にいえる。

6. 部分計算によるシステムの効率化

図 6は、幾つかのプログラムの実行速度を比較したものである。比較には20の異なる数式を用い、それぞれの速度比を平均した。

プログラムp1は、メタインタプリタ＋オブジェクト

```
a :- b, c.
a :- d, e.
b :- b1, !, b2
b :- b3.
d :- d1.
d :- d2.
```

(1)

```
a :- b1, !, b2, c.
a :- b3, c.
a :- d1, e.
a :- d2, e.
```

(2)

```
a :- b, c.
a :- d1, e.
a :- d2, e.
b :- b1, !, b2.
b :- b3.
```

(3)

図5　cut を含むプログラムの部分計算

	p1	p2	p3	p4
Rate of execution speed	100%	57.3%	55.8%	52.4%
Readability, Maintainability	○	○	○	×
execution speed	slow	fast	fast	fast

P1: Meta-interpreter + Object program
P2: Specialized program (First argument is World name)
P3: Specialized program (First argument is Goal)
P4: Object program alone

図6　実行速度の比較

プログラムであり、メタプログラミング技法によるオリジナルプログラムである。p2は、メタインタプリタにオブジェクトプログラムを与えて特殊化されたプログラムである。ここで、メタインタプリタの引数は、domain world,goal,cutmark の順である。p3は、p2と同様に部分計算によって特殊化されたプログラムであるが、メタインタプリタの引数は、goal,domain world,cutmark の順である。p4は、メタインタプリタを用いなくとも動作するように、オブジェクトプログラムのdomain world名を取り去るなどの人間によるハンドコンパイルを施したプログラムである。

p1は、プログラムの読みやすさ、修正のしやすさなどの特徴を持つ代りに、実行速度が犠牲になっている。p4は、それとは対照的な特徴を持っている。p2とp3は、オリジナルプログラムの持っているプログラムの読みやすさ、修正のしやすさなどを犠牲にしないで、実行速度の改善を行っている。つまり、プログラムの修正はオリジナルプログラムで行い、実行は特殊化されたプログラムで行われるからである。

また、p2とp3の差であるが、これはDEC-10 Prologコンパイラが各節のヘッドの第1引数のprincipal fanctor に対してindexingするからである。このシステムでは、principal fanctor の種類がgoalは22に対し、domain world名は 8である。このシステム場合には、goalを第1引数に持ってきた方が実行効率が上がるのである。

7. おわりに

本論文では、メタプログラミングによる数式処理システムへの部分計算の応用について論じた。メタプログラミング技法は、その表現力の強さによってlogic programming の中で重要な役割を果しつつある。そして、部分計算はこのメタプログラムの実行効率を高めることを可能にし、メタプログラミング技法をより効果的に用いることができるようになった。

今後の研究としては、数式処理システムの並列論理型言語（例えば、GHC [Ueda 85], CP [Shapiro 83b], PARLOG [Clark 84]）による構築を考えている。

《謝辞》　なお、本研究の機会を与えて下さったICOT研究所　渕一博所長、および、有益な討論をいただいたICOT第1研究室の各氏に深謝いたします。

参考文献

[Bundy 81] Bundy,A. and Welham,B.: Using Meta-level Inference for Selective Aapplication of Multiple Rewrite Rule Sets in Algebraic Manipulation, Artificial Intelligence No.16, pp.189-212 (1981).

[Bowen 81] Bowen,A. and Kowalski,R.: Amalganating Language and Meta-language in Logic Programming, June (1981).

[Clark 84] Clark,K. and Gregory,S. : PARLOG: Parallel Programming in Logic, Research Report DOC84/4, Imperial College (1984).

[Furukawa 84] Furukawa,K., Kunifuji,S., Takeuchi,A. and Ueda,K.: The Conceptual Specification of the Kernel Language Version 1, ICOT TR-052 (1984).

[Shapiro 83a] Shapiro,E. : Algorithmic Program Debugging, The MIT Press (1983).

[Shapiro 83b] Shapiro,H. : A subset of Concurrent Prolog and its Interpreter, ICOT TR-003

[Takeuchi 85a]竹内彰一、近藤浩康、大木健、古川康一：部分計算のメタプログラミングへの応用、情報処理学会ソフトウェア基礎論研究会(1985).

[Takeuchi 85b]竹内彰一、古川康一：Prologプログラムの部分計算とメタプログラムの特殊化への応用、Proc. of the Logic Programming Conference'85, ICOT (1985).

[Takewaki 84] 武脇敏晃、宮地泰造、阪部進、古川康一：Prologによる数式処理システム試作の構想、日本ソフトウェア科学会第1回論文集、1B-4 (1984).

[Takewaki 85] Takewaki,T., Miyachi,I. Kunifuji,S. and Furukawa,K. : An Algebraic Manipulation System Using Meta-level Inference Based on Human Heuristics, to appear in ICOT TR (1985).

[Ueda 85] Ueda,K. : Guarded Horn Clauses, Proc. of the Logic Programming Conference'85, ICOT (1985).