ICOT Technical Memorandom: TM-0961

TM-0961

項書き換えシステムから並列論理型言語への変換

吉田　和樹、大須賀　昭彦、木位田　真一、
楠井　洋一（東芝）

October, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

項書き換えシステムから並列論理型言語への変換
A PROGRAM TRANSFORMATION FROM TERM REWRITING SYSTEMS INTO PARALLEL LOGIC PROGRAMS

吉田 和樹　　大須賀 昭彦　　本位田 真一　　楠井 洋一
KAZUKI YOSHIDA　AKIHIKO OHSUGA　SHINICHI HONIDEN　YOUICHI KUSUI

（株）東芝　システム・ソフトウェア技術研究所
TOSHIBA SYSTEMS & SOFTWARE ENGINEERING LABORATORY

本稿では、項書き換えシステムから並列論理型言語ＫＬ１への変換アルゴリズムについて述べる。このアルゴリズムから得られるＫＬ１プログラムの実行結果は、項書き換えシステムで最内戦略に基づいて書き換えを行った結果得られる既約項と一致する。さらに、最内戦略が停止しない場合でも、既約項が存在すればうまくこれを獲得することができる。

１．はじめに

ソフトウェアの複雑化、大規模化にともない、システム開発後期において要求とのくい違いが表面化し、その結果、設計の見直し、コーディングのやり直しが生じるといった事態が増加する傾向にある中で、形式的仕様記述をもとにした新しいプログラミング・パラダイムの構想が浮かび上がってきた。すなわち、仕様を形式的に作成し、それにともなう形式的な手法を用いて開発初期にこれを十分に検証し、要求とのくい違いがないことを確認したうえで、仕様に連続的に変換を施して実行可能なコードを生成していこうというものである。

本稿では、形式的な仕様として代数的仕様に着目し、その直接実行系である項書き換えシステムから、実機で実行可能なＫＬ１への変換について考える。ＫＬ１はprologをもとにICOTが開発した並列論理型言語である[7]。

代数的仕様の変換についての他の研究成果としては、項書き換えシステムを模倣することによって代数的仕様を記号実行するシステムOBJ[2]や、項書き換え規則への展開・畳み込み変換の適用に関する研究[5]、さらに、C言語を初めとする手続き型言語を用いて代数的仕様の直接実行系をどのように作ればよいかを明らかにした酒井らの研究[6]、再帰的な等式集合をより強力な計算能力を持つ論理型プログラム（prolog）に変換するアルゴリズムを示した富樫らの研究[1]、などが挙げられる。

本研究は論理型言語を対象にしている点で、上記成果の中で特に富樫らの研究と近い関係にあるが、富樫らは逐次実行の論理型言語を考えているので、この成果を並列の場合に直接当てはめることはできない。すなわち、逐次実行の場合も並列実行の場合もアルゴリズム自体に本質的な差異はないが、最内戦略による書き換え結果と変換後生成されるプログラムの実行結果が一致することを示す証明において、並列実行では、再帰的に分解された部分項の書き換えを意味するサブゴールの簡約が、簡約可能なすべてのリテラルについて並列に進むことになるので、上位のリテラル（すなわち内側の部分項）から順次簡約を行うことを前提にした富樫らの証明を、そのままこれに用いることはできない。そこで、本研究ではゴールの変化に時間軸を導入して、その上でいくつかの証明の追加、補正を行い、これらの一致を示す。また、並列実行では、リテラルの簡約がすべて終了する前に求める答えが得られている場合がある。これは停止性を持たない項書き換えシステムにおいても、既約形が存在すればうまくこれを獲得できる可能性を示唆しており、この意味において本アルゴリズムから生成されたプログラムは最内戦略よりも強力な計算能力を持つ書き換え戦略の実現になり得る。この計算能力を生かすためには、答えが求まった時点で無限の簡約を停止させるしくみが必要である。

本稿では、紙面の都合上変換アルゴリズムの説明を中心に行う。

２．変換アルゴリズム

項書き換えシステムから並列論理型言語ＫＬ１への変換を考えるにあたり、対象の項書き換えシステムに次の前提を設ける。

［前提１］

(1) 項書き換えシステムは正則（重なりがない、左線形）である。

(2) 用いられる関数記号は、演算子と構成子に分類できる。すなわち、関数記号の集合をF、演算子の集合をD、構成子の集合をCとすると、

$F = D \cup C$ かつ $D \cap C = \phi$

である。

(3) d∈Dを含む基礎項（変数を含まない項）は可約である。

(4) $c_1,\ldots,c_n \in C$のみから構成される基礎項は既約である。

(5) 書き換える項として入力されるのは、基礎項とする。 □

(1)は理論的側面を考察する上での礎石である。(2),(3),(4)はHuet,Hullotの潜在帰納法におけるdefinition of principle[3]に対応するものである。(5)は(3)とあわせることで変換を簡素化して示すためのものである。

このような前提を設けることで、項書き換えシステムから論理式への次のような変換アルゴリズムを考えることができる。

[定義２]

書き換え規則l⇒rは、次のアルゴリズムに従って論理式に変換される。ただし、l=d(arg) (d∈D)、Vを書き換え規則の中で使われている変数の集合、rootを項の最外関数記号を返す関数とする。

```
function  trans( l:項, r:項 )
                returns  ２つの論理式
    l' := transL( l, Ans, Ter )
    r' := transR( r, Ans, Ter )
  return( l':- r'. )
end.

function  transL( t:項, X:変数, Y:変数 )
                returns  節のならび
    t' := tの引数の最後にX、Yの順で変数を加
          えたリテラル
  return(t')
end.

function  transR( t:項, X:変数, Y:変数 )
                returns  節のならび
  if( t∈V ) then
    return( X = t )
  else if( root(t)∈C ) then
      t1,...,tnを引数とする。
      x1,...,xnを未使用の変数とする。
      t':= tの各tiをxiで置き換えた項
    return( transR( t1, X1, Y ),...,
            transR( tn, Xn, Y ),
            X = t' )
  else if( root(t)∈D ) then
      t1,...,tnを引数とする。
      x1,...,xnを未使用の変数とする。
      t' := root(t)を述語記号と見なし、各tiを
            xiで置き換え、かつ、第n+1引数にX、第n+2引数にYを加えたリテラル
    return( transR( t1, X1, Y ),...,
            transR( tn, Xn, Y ),
            t' )
end.
```

□

このアルゴリズムを使った変換の例を次に示す。

[例３]

次のような項書き換えシステムを考える。

```
sort: elm, list;
opns: C = {[],cons}, D = {app,rev}
      [] : →list,
      cons : elm,list → list,
      app : list,list → list,
      rev : list → list;
rule: V_elm = {Z}, V_list = {X,Y}
      app([],X) ⇒ X,                    (1)
      app(cons(Z,X),Y) ⇒
                    cons(Z,app(X,Y)),(2)
      rev([]) ⇒ [],                     (3)
      rev(cons(Z,X)) ⇒
              app(rev(X),cons(Z,[]));(4)
end.
```

上記のアルゴリズムに従ってこれらの書き換え規則を論理式に変換すると次のようになる。

```
(1) app([],X,Ans,Ter) :- Ans = X.
(2) app(cons(Z,X),Y,Ans,Ter) :-
                          E = Z,
                          G = X,
                          H = Y,
                          app(G,H,F,Ter),
                          Ans = cons(E,F).
(3) rev([],Ans,Ter) :- Ans = [].
(4) rev(cons(Z,X),Ans,Ter) :-
                        F = X,
                        rev(F,D,Ter),
                        G = Z,
                        H = [],
                        E = cons(G,H),
                        app(D,E,Ans,Ter).
```

□

このアルゴリズムでは、書き換え規則の左辺と右辺で別の処理を行っている。

左辺については、２つの変数引数をアーギュメントの最後に加えてリテラルにしている。

右辺についてはその項を再帰的に部分項に分解して、最内項の書き換え結果を１階層上の部分項のアーギュメントに返すようにしている。その際、部分項の最外関数記号が演算子の場合には、前提

より可約なので、この演算子を述語記号と見なしそのアーギュメントに新たに２つの変数引数を加えてリテラルとし、その書き換えがサブゴールとして実行されるようになっている。書き換え結果は加えられた変数のうちの最初の変数に返される。２番目の変数の役割については、５.で述べる。最外関数記号が構成子（または、変数）の場合には、前提より既約なのでそのままの形で１階層上に返される。

そして、左辺の変換結果をヘッドに、右辺の変換結果をボディーに置いて論理式を生成しているのである。

３．正当性の証明

今、書き換えを行いたい項をtとすると、

```
:- transR( t, Ans, Ter ).
```

により、書き換えが実行される（ここで、transRは変換アルゴリズム中で定義した関数であり、節のならびを返すものである。ＫＬ１の述語ではないことを明記しておく。）。すなわち、tについても再帰的に部分項に分解して（前提よりtが変数を含んでいることはないと言えるが、含まれている場合も扱うことはできる。すなわち、この変数を構成子の定数と考えればよい）、演算子リテラルへのサブゴール展開を行い、これと書き換え規則から生成された論理式を使って、簡約により項書き換えをシミュレートするのである。そして、最終的な書き換え結果は変数Ansに代入されることになる。

ところで、任意の項について、項書き換えシステムで最内戦略に基づいて書き換えを行った結果ある項が得られたとすると、その書き換え規則をアルゴリズムに通して得られる論理式の集合を使った簡約結果もこれに一致する。つまり、次の命題が成り立つ。

［命題４］

前提１を満たす任意の項書き換えシステムＲと、その書き換え規則をアルゴリズムに通して得られる論理式の集合Ｐを考える。任意の項tについて、最内戦略による書き換え結果t'が存在するならば、

```
:-transR(t,Ans,Ter).
```

のAnsには、t'が入る。 □

４．ＫＬ１への変換

ところで、この変換アルゴリズムにより生成される論理式に、次で説明する明らかに意味不変な変換を施すとＫＬ１プログラムが得られる。

［例５］

例３(1),(4)の論理式をＫＬ１に変換する。

```
(1) app([],X,Ans,Ter) :- Ans = X.
                    ↓
    app([],X,Ans,Ter) :- true | Ans = X.

(4) rev(cons(Z,X),Ans,Ter) :-
                          F = X,
                          rev(F,D,Ter),
                          G = Z,
                          H = [],
                          E = cons(G,H),
                          app(D,E,Ans,Ter).
                    ↓
    rev(cons(Z,X),Ans,Ter) :-
                  true | rev(X,D,Ter),
                  app(D,cons(Z,[]),Ans,Ter).
```

□

この変換では、右辺部分項の最外関数記号が構成子（または、変数）の場合には、変数化されている引数に直接その項を当てはめて、無駄な単一化を減らすことが行われている。ただし、ＫＬ１のシンタックス上、ボディーでの計算結果をヘッドの変数に直接書き込むことは許されないので、Ansはそのまま残されることになる。そして、ボディーの先頭にガードtrueを加えれば、ＫＬ１のシンタックスに合致した論理プログラムが完成する。これが、アルゴリズムを通して得られた初期の論理式の集合と意味的に等価であることは自明である。

５．計算停止アルゴリズム

ここで、次のような例を考える。

［例６］

```
sort: nat,bool,list;
opns: C = {cons,[]},
      D = {gen,if,check}
      cons:nat → list,
      gen:nat → list,
      []: → list,
      if:bool,list,list → list,
      check:list → list;
rule: V_nat = {X}, V_list = {Y}
      gen(X) ⇒ cons(X,gen(X+1)),
      check(cons(X,Y)) ⇒
            if(X==3,[],cons(X,check(Y)));
end.                                        □
```

ここで、+や-については通常の意味で定義済みであることを仮定する。

この書き換え規則からＫＬ１プログラムを生成して、check(gen(0))の書き換えを並列実行により行う場合、書き換え指示を

```
:- transR(check(gen(0)),Ans,Ter).
```

すなわち、

```
:- B = 0,
   gen(B,A,Ter),
   check(A,Ans,Ter).
```

のみで与えると、genとcheckのresolutionが同時に起こり、変数Ansには書き換え結果

```
cons(0,cons(1,cons(2,[])))
```

が返されるが、それ以降もgen(B,A)の簡約は無限に続くことになる。このように、項書き換えシステムが停止性を持たない場合でも既約形を求めることができる本プログラムの計算能力を考慮すると、無限の簡約を停止させる次のような仕組みをさらに加える必要がある。（ここで、演算子のアーギュメントに新たに加えた２番目の変数が大きな意味を持つことになる。） すなわち、

(1)任意の演算子d∈Dについて、次の節を加える。

```
d(_,_,Ans,stop) :- true | Ans = [].
```

（演算子dの引数を_で置き換える。上はdのarityが２の場合）

(2)書き換えを指示するゴール節を次のように与える。

```
:- transR( t, Ans, Ter ),
   nf( Ans, stop, Ter ).
```

nfは、変数Ansに書き換え結果が返されたならば、変数Terにstopを代入する述語である。そのコーディングは、次のようになる。

```
nf(T,Fh,Ft) :- atom(T) | Fh = Ft.
nf(T,Fh,Ft) :-
        vector(T,L) | nf_args(T,1,L,Fh,Ft).
nf_args(T,L,L,Fh,Ft) :- true | Fh = Ft.
nf_args(T,P,L,Fh,Ft) :-
                    P(L,
                    vector_element(T,P,S) |
                    nf(S,Fh,Fm), P1 := P+1,
                    nf_args(T,P1,L,Fm,Ft).
```

変数Terはすべての演算子リテラルが共有しているグローバル変数であり、ここにstopが代入されれば、無限の簡約を起こしているリテラルと上記(1)で追加した節とのパターンマッチにより、書き換え結果を収納する変数に[]が代入されて、無限の簡約は停止する。

6．今後の課題

本研究では、並列論理型言語を対象にした変換アルゴリズムを示し、これと最内戦略による項書き換え結果の関係を明かにした。

ところで、本アルゴリズムから得られる論理プログラムの並列実行は、５．の例題でも挙げたように、書き換え規則が停止性の条件を満たさなくても答えがあるならば必ずそれを獲得する、という意味で正規化戦略の実現になっていると考えられる。

この観点から、今後はより強力な計算能力を持つ書き換え戦略と本アルゴリズムの関係について考察していく予定である。

7．謝辞

本研究は、第５世代コンピュータプロジェクト(FGCS)の一環として行われたものである。研究の機会を与えてくださった、ICOT 淵一博所長、(株)東芝 システム・ソフトウェア技術研究所 西島誠一所長、大筆豊部長に深く感謝いたします。

参考文献

[1]Togashi, A. and Noguchi, S. : A Program Transformation from Equational Programs, The Journal of Logic Programming, vol.4, 1987, pp.85-103.

[2]二木厚吉、中川中：抽象データ型とOBJ2、bit, Vol.20, No.9, 1988, pp.1037-1050.

[3]Huet, G. and Hullot, J. M. : Proofs by Induction in Equational Theories with Constructors, Journal of Computer System Science, vol.25, 1982, pp.239-266.

[4]井田哲夫：新しいプログラミング・パラダイム、共立出版, 1989.

[5]Darlington, J. : The Synthesis of Implementations for Abstract Data Types, A Program Transformation Tactic, Computer Program Synthesis Methodologies, D. Reidel Publishing Company, 1983, pp.309-334.

[6]酒井正彦、坂部俊樹、稲垣康善：抽象データ型の代数的仕様の直接実現系Cdimple、コンピュータソフトウェア、Vol.4, No.4, 1987, pp.16-27.

[7]新世代コンピュータ技術開発機構　第四研究室：ＫＬ１プログラミング　入門編・初級編・中級編、1989.