ソフトウェア基礎論　17－12
(1986. 7. 4)

万能項書き換えシステムと部分計算

A UNIVERSAL TERM REWRITING SYSTEM AND PARTIAL COMPUTATION

山中英樹+　　直井　徹+　　坂部俊樹++　　稲垣康善+

Hideki YAMANAKA　Tohru NAOI　Toshiki SAKABE　Yasuyoshi INAGAKI

+ 名古屋大学　工学部　　++ 三重大学　工学部

+ Nagoya University　　++ Mie University

あらまし　　本稿では，項書き換えシステムによって表現される計算の過程は，部分計算の過程として特徴付けられることを示し，その観点に基づいた射影機械の項書き換えシステムによる実現の試みを報告する．また，その射影機械の言語開発支援システムへの応用について述べる．

Abstruct　In this paper, we show that computation process of term rewriting system can be characterized as partial computation. We construct a projection machine as a term rewriting system. We try to use the projection machine to generate a code optimization phase of "Lass" (Language Development supporting System).

1．はじめに

項書き換えシステムは，ソフトウェアの代数的仕様記述に操作的意味論を与えるのに用いられたり，また，定理証明の道具としてもその有効性が注目されている．それればかりでなく，新しい計算のモデルとしても脚光を浴びており，理論，実用の両面から項書き換えシステムの重要性の認識が高まっている．

一方，プログラムの部分計算の理論は，そのコンパイラ生成系へのアプローチのユニークさで有名であり，部分計算プログラムあるいは射影機械の実現に向けての研究が盛んである．

本稿では，項書き換えシステムによって表現される計算の過程は，部分計算の過程として特徴付けられることを示し，その観点に基づいた射影機械の項書き換えシステムによる実現の試みを報告する．また，その射影機械の言語開発支援システムへの応用について述べる．

2．項書き換えシステム

この節では，本稿で必要な範囲で項書き換えシステムの基本概念と性質について説明する．

定義　1．項書き換えシステムは，次の性質を持つ項集合上の二項関係 R である．

(1)　$\xi == \eta \in R$のとき，ξ は変数ではない．

(2)　$\xi == \eta \in R$のとき，η に現れる変数は ξ に現れていなければならない．

Rの要素は，書き換えルール（又は単にルール）と呼ばれる．

定義　2．項Mの部分項で，ルールの左辺中の変数に適当な項を代入して得られる項と一致する場合に，その項をリデックスと言う．項 M のリデックスをルールに従って書き換えて項 N が得られたならば，M は N にリダクションされたといい，$M \to N$ と書く．また，M から0回以上のリダクションで N に到達可能であるとき

$M \twoheadrightarrow N$ と書く．さらに，項 N がリデックスを持たないならば，N は正規形 (normal form) であるという．

定義　3．項書き換えシステムが線形であるとは，どのルールの左辺にも同じ変数が二度以上出現しないことを言う．

定義 4. 項書き換えシステムが無あいまいであるとは，二つの異なるどのルール x==x'，y==y' においても，x の部分項 z が y と単一化可能（unifiable）でないことをいう．

定義 5. 項書き換えシステムが合流性をもつとは，任意の項 x，y，z において，x⇒y，x⇒z ならば，適当な項 w が存在して y⇒w，z⇒w とできることをいう．

合流性をもつ項書き換えシステムでは，ある項を適当に書き換えて行って正規形に到達すれば，その正規形以外にその項の正規形は存在しないことが保証される．

合流性を満たすための構文上の制限として，次の定理はよく知られたものである．

定理 1. 項書き換えシステムが線形かつ無あいまいであるとき，項書き換えシステムは合流性をもつ．

項 M は一般に複数のリデックスをもつので，書き換えるリデックスの選び方により，リダクションが異なることがある．従って，リダクションの各ステップで書き換えるべきリデックスを決める規則が必要であり，その規則をリダクションの戦略という．項Mに正規形が存在しても，リダクションの戦略によっては正規形に到達できないことがある．

定義 6. 任意の項Mに対して，Mに正規形が存在するならば必ず正規形に到達できる戦略を正規化戦略と言う．

無あいまい線形な項書き換えシステムについては次のことが知られている．

定義 7. 他のリデックスの部分項になっていないリデックスを最外リデックスと言う．項Mに正規形が存在するならば，最外リデックスのすべてを常に書き換えるという戦略は，並列最外戦略（parallel outer-most reduction）と呼ばれる．

定理 2. 無あいまい線形な項書き換えシステムに対しては，並列最外戦略は，正規化戦略である．

本稿では，扱う項書き換えシステムは，すべて線形でかつ合流性を持つと仮定する．

3．項書き換えシステムにおける部分計算（混合計算）

項書き換えシステムは，そのルールに従って，与えられた初期項を可能な限り書き換え，最終的に得られる項を出力する働きをする．従って，項書き換えシステムと初期項との組みはある意味での計算の過程を表現するものであり，一種のプログラムであると考えられる．このプログラムへの入力は，初期項に含まれる自由変数に基礎項（変数を含まない項）を代入することで行われるとみなすのが自然であろう．

このような項書き換えシステムと初期項の対で表される計算は，入力データが完全に与えられなくても支障なく計算が進められ，また，部分的な入力データによって得られた計算結果に残りの入力データを与えて計算を進めた結果と，初めから全入力データを与えて計算した結果が一致する．この意味で，項書き換えシステムと初期項の表す計算は部分計算であると言うことができる．さらに，項書き換えシステムと初期項の対であるプログラムを解釈実行する万能プログラム（これも項書き換えシステムと初期項の対）を考えると，この万能プログラムは，明らかに，二村[4]，Ershov[3]の言う部分計算（混合計算）に対応している．このことを以下に明らかにする．

Ershov に従えば，プログラムの部分計算は次のように言うことができる．（ただし，理解し易いように少し変えてあるところがある．）

入出力変数へのデータの割り当ての集合 D，プログラムの集合 P，プログラムに意味を与える写像 V：P×D −> D とからなる三項組 L=(D,P,V) をアルゴリズム言語（以下では単に言語）と言う．このとき，言語Lにおける射影機械は次の (1),(2), (3) を満たす写像

$\alpha : P\times D\times D \rightarrow P\times D$

である．

任意の $p\in P$，$d,\bar{d}\in D$ に対して，

(1) $\alpha(p,d,\phi) = (\phi, V(p,d))$

(2) $\alpha(p,\phi,d) = (p,d)$

(3) $V(p,d\cup\bar{d}) = V(\alpha(p,d,\bar{d}))$

ただし，αの第二引数と第三引数は，その和が矛盾なくすべての入力変数にデータを割り当てるような割り当てであるとする．

これに対して．項書き換えシステムと項との対がプログラムであるような言語Tは，上の形式に合わせると T=(S,P,V) となる．ここに，

$S = \{\sigma : X \to T[\Sigma] \mid \Sigma$はシグニチャ$\}$

$P = \{<R, \xi> \mid R$は項書き換えシステム，ξは項$\}$

$V : P \times S \to S$

$V((R,\xi),d) = \{y \leftarrow NF_R(\sigma(\xi))\}$

である．ただし，Xは出力変数yを含む自由変数の集合で，すべてのプログラムについて共通であるとする．また，$NF_R(\sigma(\xi))$は，ξにσによる代入を施した項$\sigma(\xi)$をRで書き換えて得られる正規形を表し，

$\{y \leftarrow NF_R(\sigma(\xi))\}$

は，yに$NF_R(\sigma(\xi))$を割り当てる代入である．

任意の項書き換えシステムRと項ξに対して，そのプログラム（R，ξ）をシミュレートするプログラムUで，次の条件

$$U((R,\xi),\sigma,\bar{\sigma}) = ((R, NF_R(\sigma(\xi))),\bar{\sigma})$$

が成立するとき，このようなプログラムUを万能プログラムと呼ぶ．Rは，合流性を持つと仮定しているので明らかに次の(4),(5),(6)が成り立つ．

(4) $U((R,\xi),\sigma,\phi) = (\phi, (V(R,\xi),\sigma))$

(5) $U((R,\xi),\phi,\sigma) \sim ((R,\xi),\sigma)$

(6) $V((p,\xi),\sigma) = V(U((R,\xi),\sigma,\bar{\sigma})$

ただし，条件(5)の～は

$((R,\xi),\sigma) \sim ((R',\xi'),\sigma')$ iff

$R=R'$，$\sigma=\sigma'$，$\xi \overset{*}{\Leftrightarrow} \xi'$で定められる関係である．(5)の中の$\xi$が正規形であるとすれば，～は＝で置き換えることができる．

条件(4),(5),(6)は，それぞれ条件(1),(2),(3)に対応するものであり，UはTにおける射影機械になっていることが示された．

4．万能項書き換えシステム

前節で述べた万能プログラムUを項書き換えシステムと項の対で実現したので，本節ではそれについて述べる．煩雑さを避けるため，Uにおける，項書き換えシステムの部分（これを万能項書き換えシステムと言う）の記述についてのみ述べることにする．

先ず，万能項書き換えシステムの作成法を直感的に例をもって説明し，それから具体的に記述法を説明する．

例えば，LISPにはM式とS式と呼ばれる表現がある．LISPは，リストの操作をするための言語であり，S式は，リスト型のデータを表す記法であるので，データがリストの型（S式）で与えられれば，それを処理するプログラムを書くことができる．M式を対応するS式に変換（coding）する方法を定め，S式で表されたプログラムを処理するプログラムをM式で書けば，M式で書くことができるすべてのプログラムを処理するプログラムを書いたことになる．このようなプログラムを実行するための関数は万能関数（universal function）とよばれる．

同様に，項書き換えシステムは，項を操作するためのシステムであるので，データが項の形で与えられれば，そのデータは項書き換えシステムで処理できる．M式をS式に coding したのと同様に，項書き換えシステムのプログラムを coding して項の形に表現すれば，それを入力としてそれが表す動作をシミュレートする項書き換えシステム（万能項書き換えシステム）を構成でき，これでもって任意の項書き換えシステムのプログラムを計算できる．

以下に，我々の構成した万能項書き換えシステム UTRS について説明する．そのために，UTRS への入力である項書き換えシステムと次の coding ならびに，その coding された項書き換えシステムと項の対に対して UTRS がどのように動作するか明らかにする．

先ず，項ならびに項書き換えシステムの coding について述べる．これを形式的に述べることは手数だけがかかることであるので，例を用いて簡単に説明する．例えば，f(g(x),y) という項は，LISP のS式に coding すると (f (g x) y) となる．次にこれを万能項書き換えシステムの内部表現として次のように coding する．

```
CONS_TERM( FUNC(f),
  CONS_TERM-LIST(
    CONS_TERM( FUNC(g),
      CONS_TERM-LIST( VAR(x),NIL_TERM-LIST())),
    CONS_TERM-LIST( VAR(y), NIL_TERM-LIST()))
```

ルールは上で述べた coding された項の対に，また，書き換えルールの集合はルールのリストに coding する．そうすると，書き換えルールの集合は下のようなリストになる．

```
CONS_RULE-LIST(PAIR_TERM(t1,t2),
  CONS_RULE-LIST(PAIR_TERM(t3,t4),
                 ...,NIL_RULE-LIST())
```

ただし，t1,t2,t3,t4,...は coding された項を表す．簡単に，coding された後の項の構文の生成規則を書くと次のようになる．

```
<term>::=CONS_TERM(FUNC(<symbol>),<term-list>)
      ::=VAR(<symbol>)
<term-list>::=CONS_TERM-LIST(<term>,<term-list>)
           ::=NIL_TERM-LIST()
<rule-list>::=CONS_RULE-LIST(<rule>,<rule-list>)
           ::=NIL_RULE-LIST()
<rule>::=PAIR_TERM(<term>,<term>)
```

また，必要であれば，内部表現になっている計算結果を逆変換（decoding）し，元の項書き換えシステムの項に戻すこともできる．

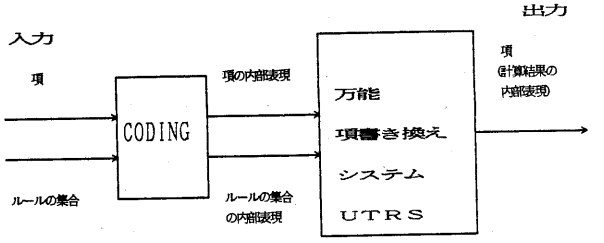


Fig 1. 万能項書き換えシステム UTRS の概念図

次に，具体的に万能項書き換えシステム UTRS の作成方法を説明する．直観的には，上述のように coding された項に対して適用可能なやはり coding された書き換えルールを探してそれを適用することが UTRS の動作である．以下では，特に考慮した，(a) 正規形の判定（書き換え停止の判定），(b) 正規化戦略，(c) 実現に依存しない記述法，について説明する．

(a) 先ず，項を書き換えようとする操作を停止させるためには，項が正規形に到達したことを判定できなければならない．しかし，項書き換えシステムでは書き換えが局所的に行われるので，直接的に項全体が正規形であるかどうかという大域的な判定を下すことはできない．そうすると，部分項が正規形であるという局所的な情報を合成して，項全体が正規形であるという大域的な情報を作り出す操作が必要になる．そこでどうするかというと，UTRS では項をリストとして表すためのの構成子を導入しているので，ついでにこの構成子にタグの機能を持たせることを考える．つまり，構成子に二種類のもの，正規形である項の構成子 NORMAL_TERM と正規形であるかどうか分かっていない一般の項の構成子 CONS_TERM を考える．そして，書き換え操作の途中である部分項が正規形であることがわかったら，その部分項の構成子を CONS_TERMから NORMAL_TERM へ換えていくことによって，正規形である部分項をより集めてより大きな正規形の部分項を作って行く．そうして，書き換えるべき項自体の最外の構成子が NORMAL_TERM になったとき，この項が正規形に到達したと判定する．このことを記述するために次の関数記号 APPLY_RULE-LIST とそれに対する次のルールを導入する．

```
APPLY_RULE-LIST(
        CONS_RULE-LIST(rule1,rule-list1),
        POSSIBLY-REDEX(
                CONS_TERM(
                        f-symbol1,
                        term-list1)),
        rule-list2)
== APPLY_RULE-LIST(
        rule-list1,
        TRY_REWRITE_TERM(
                rule1,
                POSSIBLY-REDEX(
                    CONS_TERM(
                        f-symbol1,
                        term-list1))),
        rule-list2) ;
```

APPLY_RULE-LIST の領域は，次のよう定められている．

```
APPLY_RULE-LIST : rule-list,cand-redex,
                        rule-list --> term ;
```

ここで，rule-list を二つ引数にもつのは，第一引数の rule-list は，マッチングするかどうか調べて行く過程で，一つずつ削ってより小さなリストにしていくためのものであり，第三引数の rule-list は，項が評価されるべき環境である．この上のルールには，APPLY_RULE-LIST が書き換えルールのリストの要素を順番に取り出して，これと項がマッチするかどうか確かめてマッチしていれば項を書き換えることが記述されている．この関数記号を用いて上で述べた正規形の項からより大きな正規形の項を作るルールは，次のように記述できる．

```
APPLY_RULE-LIST(
    NIL_RULE-LIST(),
    POSSIBLY-REDEX(
      CONS_TERM(
          FUNC(symbol1),
          NORMAL_TERM-LIST(
                term1,
                term-list1))),
    rule-list1)
== NORMAL_TERM(
      FUNC(symbol1),
      NORMAL_TERM-LIST(term1,term-list1)) ;
```

また，このルールは，リデックスかもしれない項(cand-redex)に対しルールのリストの要素がマッチングするかどうか順番に試して行き，結局マッチするルールが無かったとき，しかも，その項の部分項がすべて正規形であったときは，リデックスの候補であったその項は実は正規形であるという性質を表している．

(b) 次に，UTRS の正規化戦略について説明する．あるプログラムを全計算すると計算が停止するが，部分計算すると計算が停止しないことがある．そんな場合でも，リダクションの戦略によっては，部分計算しても停止する場合がある．例えば，

```
F(x,y) == SI( x≧0,x+y,F(x+1,y+1)) ;
SI(true,x,y)==x ;
SI(false,x,y)==y ;
```

という関数 F,SI が与えられたとき，F(2,y) を最左最内書き換え (call by value) で部分計算すると，

```
F(2,y) → SI( 2≧0, 2+y, F(3,y+1))
→ SI( true,2+y,SI(3≧0,4+y,F(4,y+2)))
→ SI( true,2+y,SI(true,4+y,SI(4,6+y,SI(...))))
```

となり，計算は停止しない．しかし，最外リデックスを書き換える戦略をとると 2+y という計算結果を得ることができる．このように，計算が停止する戦略をとることは重要である．現在，正規化戦略として理論的に最も強力であると考えられるものに並列最外リダクションがある．そこで，UTRS の正規化戦略を並列最外リダクションに選んだ．これは，次のようなリダクションである．先ず，関数記号 EVAL_STEP を導入し項を one-step だけリダクションするルールを考える．

```
EVAL-STEP(
          NORMAL_TERM(
                f-symbol1,
                term-list1),
          rule-list1)
 == NORMAL_TERM(
          f-symbol1,
          term-list1) ;
```

上のルールは，正規形の項を one-step リダクションしようとしてもそのままであることを表している．

```
EVAL-STEP(
          CONS_TERM(f-symbol1,term-list1),
          rule-list1)
 == APPLY_RULE-LIST(
          rule-list1,
          POSSIBLY-REDEX(
                CONS_TERM(
                      f-symbol1,
                      term-list1)),
          rule-list1) ;
```

このルールは，正規形であると分かっていない項はリデックスである可能性があるので，APPLY_RULE-LIST によってルールとのマッチングを調べるためのものである．APPLY_RULE-LIST は，先ほど説明したようにルールのリストを一つずつ削ってそのルールとリデックスの候補になっている項とのマッチングをとっていくときに使われる関数記号であるが，ルールのすべてが項とマッチングしなかったときどうするかというのが次のルールである．

```
APPLY_RULE-LIST(
    NIL_RULE-LIST(),
    POSSIBLY-REDEX(
        CONS_TERM(
            f-symbol1,
            CONS_TERM-LIST(
                  term1,
                  term-list1))),
    rule-list1)
== CONS_TERM(
    f-symbol1,
    LIST_EVAL-STEP(
        CONS_TERM-LIST(term1,term-list1),
    rule-list1)) ;
```

つまり，このルールはリデックスの候補に上げていた項が結局リデックスでなかったことが分かったとき，この項の引数に対し並列に one-step のリダクションを行うものである．このように，one-step のリダクションを外側から進め，また，引数に対して並列にリダクションを進める．そして，この one-step のリダクションを,書き換える項が正規形になるまで繰り返せば並列最外戦略を実現できる．

(c) 次に，この one-step のリダクションを，書き換えるべき項が正規形になるまで繰り返す関数を記述する．この関数を EVAL_TERM とし，この関数を簡単に記述すると次のようになる．

```
   EVAL_TERM(term1,rule-list1)
== IF(
       NORMAL_P(term1),
       term1,
       EVAL_TERM(
              EVAL_STEP(term1,rule-list1),
              rule-list1)) ;

   NORMAL_P(NORMAL_TERM(term1,term-list1))
== TRUE() ;

   NORMAL_P(CONS_TERM(term1,term-list1))
== FALSE() ;
```

しかし，このままでは UTRS を実現しているシステムの戦略によっては入力された項の正規形が求められないことがある．例えば，最左最内書き換え (call by value) で NORMAL_TERM(term2,term-list2) を書き換えると，

```
EVAL_TERM(NORMAL_TERM(term2,term-list2),
       rule-list1)

→ IF(NORMAL_P(NORMAL_TERM(term2,term-list2)),
     NORMAL(term2,term-list2),
     EVAL_TERM(EVAL_STEP(
      NORMAL_TERM(term2,term-list2),rule-list1),
      rule-list1))

→ IF(TRUE(),NORMAL(term2,term-list2),
     IF(TRUE(),NORMAL(term2,term-list2),
       IF(TRUE(),NORMAL(term2,term-list2),
         ... )))
```

のように，EVAL_TERM によってリデックスが生成され続けてリダクションが停止しない．そこで次のようにルールを変える．

```
   EVAL_TERM(term1,rule-list1)
== NORMALIZE(
       EVAL-STEP(term1,rule-list1),
       rule-list1) ;

   NORMALIZE(
       NORMAL_TERM(f-symbol1,term-list1),
       rule-list1)
== NORMAL_TERM(f-symbol1,term-list1) ;

   NORMALIZE(
       CONS_TERM(f-symbol1,term-list1),
       rule-list1)
== EVAL_TERM(
       CONS_TERM(f-symbol1,term-list1),
       rule-list1) ;
```

ここで，新しく NORMALIZE を導入した．上の例で示したように EVAL_TERM による直接の再帰を用いるとリデックスが複数個できてしまい，リダクションが停止しなくなる．そこで EVAL_TERM と同じような機能を持つ関数記号 NORMALIZE を導入し，EVAL_TERM と NORMALIZE が相互に呼び合うかたちで再帰を書き，リデックスが一つしかできないようにする．こうするとリデックスが一つしかできないので，UTRS を実現するシステムの戦略に依存しない形で，しかも，並列最外戦略が記述できる．

5．万能項書き換えシステム (UTRS) の応用

5.1 インタプリタの記述からのコンパイラ自動生成

射影機械（部分計算をその計算原理とする万能計算システム）をを用いて，インタプリタからコンパイラを自動生成する方法が，二村[4]，Ershov[3]，他により提案されている．この方法の原理は，射影機械に射影機械のプログラムと対象とする言語のインタプリタプログラムを入力すると，対象言語から射影機械の実現言語へのコンパイラが得られるというものである．これを簡単に説明する．

コンパイラのソース言語は SL=(SP,SD,INT)，ターゲット言語は L=(P,D,V) とし，コンパイラを実現する言語はターゲット言語と同じＬであるとする．また，INT(x,y) は，ＳＬのインタプリタであり，Ｌのプログラムとして記述されているとする．このとき，Ｌ

の射影機械 mix(u,v,w) がＬで記述されているとすると，次の関係式が成り立つことが知られている．任意の π∈ＳＬ 関して，

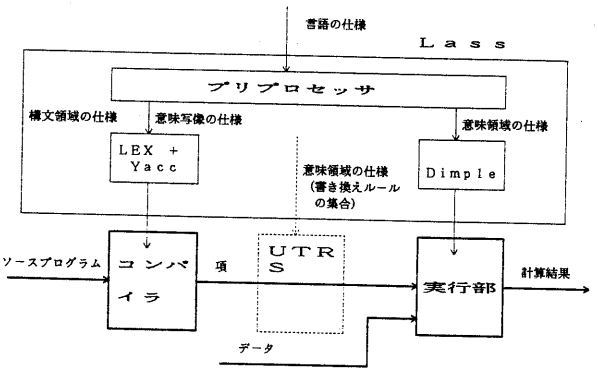
```
mix(int,{x <- π},{y <- d})=π(d)

mix(mix,{u <- int},{v <- π,w <- d})
= comp(π,d)

mix(mix,{u <- mix},{v <- {u <- int},
    w <- {v <- π,w <- d}})
= cocomp(int,π,d)
```

Ｌとして第三節で述べた，項書き換えシステムと項の対をプログラムとする言語 T=(S,P,V) を取り，mix として第６節の万能項書き換えシステムから得られる万能プログラム Ｕ を取れば，上の原理によって Ｕ を用いたコンパイラ生成系を得ることができる．

現在，UTRS には，自由変数に矛盾なくデータを割り当てる機能がない．また，自由変数の名前が局所変数の名前と衝突すると，計算の正しさが保証できない．つまり，変数名を管理する機能がないので，UTRS はまだ mix （射影機械）であるとは言えない．

5.2 コンパイラの最適化フェーズへの応用

我々は，既に言語開発支援システム Lass[5] を作成し，この Lass により手続き型言語 PL/0.2[6] の仕様記述からその言語処理系を自動生成している．この処理系の出力する目的コードは，PL/0.2 のプログラムに対応する項である．



Ｆｉｇ ２． Ｌａｓｓの生成する処理系の最適化フェーズへのＵＴＲＳの応用

Lass の生成した言語処理系を用い PL/0.2 のプログラムから変換された項（目的コード）には，入力を待たず計算できる部分がある．この部分を UTRS で前もって部分計算しておくと，実際に項にデータを与えて計算するとき，部分計算しなかったものに比べ部分計算されたものは，計算が早く済むと考えられる．

現在，我々は，Fig 2. の Lass を使い，PL/0.2 のソースプログラムをいったん項に変換してから，この項と PL/0.2 の意味領域の仕様（項書き換えシステム）を coding して，それを UTRS に入力して部分計算する実験をしている．

６．まとめ

項書き換えシステムと初期項の対をプログラムとみなすという立場に立って，プログラムの部分計算とその応用について述べた．すなわち，項書き換えシステムと項というプログラムで射影機械の本質的な部分である万能項書き換えシステムを実現し，それをコンパイラの最適化フェーズに応用した例について報告した．

万能項書き換えシステムから本来の目的である射影機械を実現し，コンパイラ・コンパイラへ応用することなどは今後の課題である．

謝辞　日頃御指導賜る豊橋技術科学大学本多波雄学長、名古屋大学福村晃夫教授、並びに御討論下さる阿曽弘具助教授をはじめとする研究室の皆様に感謝する．
なお，本研究は一部，文部省科研費（一般(ｃ)課題番号 60550263 及び，特定研究(1) 多元知識情報　課題番号　61102003 ）の援助を受けたので，記して感謝する．

参考文献

(1) 二木，他 ”項書き換え型計算モデルとその応用”，情報処理　Vol. 24 No. 2 Feb,(1983)
(2) 川辺，他 ”抽象データ型の直接実現システム”，信学技報 AL 83-65,(1984)
(3) A.P.Ershov "MIXED COMPUTATION: POTENTIAL APPLICATIONS AND PROBLEMS FOR STUDY" Theor. Comput. Sci 18 (1982) 41-67
(4) 二村 ” Partial Computation of Programs" LNCS No. 147,(1983)
(5) 酒井，他 ”プログラミング言語の代数的仕様記 述からのコンパイラ自動生成" 信学技報 AL 85-10 (1985)
(6) 高木，他 ”人出力機能およびパラメタ付手続きを付加したプログラミング言語 PL/0 の代数的仕様記述" 信学技報 AL 85-44 (1985)