6G−3

遅延ナローイングに基づく言語Evの等式翻訳方法

鈴木太朗　井田哲雄
筑波大学

1　はじめに

ナローイングは等式に関する推論を効率良く行なう方法として考えられたものであるが、最近、関数型言語と論理型言語を融合する計算機構として注目されている。また、遅延評価は、プログラミングにおけるその有効性が関数型言語の研究において広く認識されている。

現在我々は、遅延ナローイング計算系(以降LNCと呼ぶ)に興味を持っている。これは、等式の反駁による証明を計算の機構とし、証明の過程で値が必要になった関数項のみをナローイングにより簡約する計算系である。

我々は、LNCに基づく言語Evを定義した[1]。ナローイングに基づく言語を実現するには、等式の翻訳を効率よく実現することが不可欠である。本稿では、言語Evにおける等式の翻訳方法について述べる。

2　言語Ev

言語Evは、条件付き等式とゴール式より成る。条件付き等式は、

$$f(d_1,\ldots,d_n)=t \Leftarrow s_1=k_1,\ldots,s_m=k_m \quad (1)$$

ここで、$n\geq 0, m\geq 0$、

の形式で与えられる。式(1)の$\Leftarrow$の左側を条件付き等式の等式部、右側を条件部という。

ゴール式は

$$\Leftarrow s_1=d_1,\ldots,s_l=d_l \quad (2)$$

ここで、$l\geq 1$、

の形式で与えられる。

Evの式はあらかじめ基本式に変換されてから翻訳される。基本式は、以下のような性質を持つ式である。

- 条件付き等式の等式部の左辺の引数は相異なる変数
- 条件付き等式の条件部の等式の右辺は変数あるいは真部分項が変数である構成子項。
- ゴール式の等式の右辺は関数を含まない構成子項あるいは変数。

以下では、真部分項がすべて変数である構成子項を浅い構成子項、関数項をパラメータに含まない構成子項をデータ項と呼ぶ。また、基本式に変換された条件付き等式を基本条件付き等式、ゴール式を基本ゴール式と呼ぶ。

3　遅延ナローイング計算系LNC

LNCは以下の4つの推論規則を持つ。

[ln]　遅延ナローイング
[vd]　データ項の束縛
[cv]　構成子項の束縛
[u]　単一化

LNCにおける計算とは、基本ゴール式$\Leftarrow E^*$に上の推論規則を繰り返し適用してゴール式$\Leftarrow \square$を得る反駁である。この反駁で得られる代入の合成をθとする。

推論規則の適用は等式$t=d$の両辺の型により一意に定まる。表1に両辺の型と適用される推論規則を示す。

$t\backslash d$	変数	浅い構成子項
変数	[vd]	[vd]
関数項	[ln]	[ln]
構成子項	[cv]	[u]

表1. 等式の両辺の型と適用される推論規則

4　言語Evにおける等式の翻訳

言語Evにおける等式は、その両辺の型により、対応するLNCの推論規則を実行する命令列へと翻訳される。LNCの推論規則の実行には、単一化代入の作成など、Prologに似た計算機構が必要とされる。したがって、我々はEvを実行する抽象機械としてWAM[3]を基本とし、LNCの推論規則を実現できるように拡張したものを用いる[2]。ただし、WAMはレジスタマシンであるが、我々の抽象機械はスタックマシンである。したがって引数はスタックに積まれる。ただし、局所的なデータのやりとりのためにAというレジスタを用意する。

以下では、WAMの理解を前提として議論をすすめる。

4.1　与えられた等式の型による処理の選択

3で述べたように、LNCの計算の過程で単一化代入の合成θが得られる。こうして得られたθは残りの式に適用される。表1では、与えられた等式にこの単一化代入θを適用してから推論規則を選択する(与えられた等式という用語を使う時は、単一化代入θを適用していない等式を指している)。

しかし、変数の出現場所によっては、単一化代入θを適用しなくても静的に推論規則を選択できる場合がある。

基本条件付き等式$t=s \Leftarrow E^*, x=d, F^*$で$x=d$の左側に変数$x$が現れないとき$x$は最初の出現であるという。この場合は、$x=d$の計算の時点では単一化代入$\theta$中に$x$への束縛は存在しないので、$x$は必ず変数であることは明らかである。このような場合には、翻訳時に推論規則を選択することができる。

一方、xが2回目以降の出現であるならば、単一化代入θにxへの束縛が含まれている可能性がある。この場合はθを適用しなければ推論規則を選択することはできない。

この考察にしたがって、表1の変数の欄を変数の出現によって2つに分け、与えられた等式の両辺の型から実行すべき処理を示したものが表2である。

Compiling Method of Equations in Narrowing Based Language Ev
Taro Suzuki,Tetsuo Ida
University of Tsukuba

$t\backslash d$	変数の最初の出現	変数の2回目以降の出現	浅い構成子項
変数の最初の出現	(a)t に d をセット		
変数の2回目以降の出現	(b)$\theta(t=d)$ の型により処理を選択、実行		
関数項	(c)引数をスタックに積み、(d)関数項に対応する手続きを呼ぶ		
構成子項	(a)d に t をセット	(e)t と d を単一化	(a)d の要素に t の要素をセット

表2. 与えられた等式 $t=d$ の両辺の型による処理一覧

表2の (a)(c)(d)(e) に対応する命令を以下に示す。各命令の右側に例を示す。

(a)put 系命令	put_variable x,y
(c)push 系命令	push_variable x,y
(d)call 系命令	call $pred$
(e)get 系、unify 系命令	get_variable x,y

表2の (b) の場合には、実行時に推論規則を選択しなければならない。このために、θt の型によって多方向分岐する命令 jmp_if_function $x,label$ を追加する。この命令は、x をデレファレンスし、その結果が関数項ならばラベル $label$ へ分岐する。x をデレファレンスした結果はAレジスタへセットされる。

(b) の処理を実現するために、与えられた等式を以下のような命令列に翻訳する。

```
        jmp_if_function  x, label
        単一化を行なう命令列
        jmp              label'
label   関数項の書換えを行なう命令列
label'
```

ここで、命令 jmp $label$ はラベル $label$ への無条件分岐命令である。関数項の書換えを行なう命令列は4.2で説明する。

4.2 関数項の書換え

与えられた等式の左辺が関数項である場合には、その関数項の書換えは表2で示したように簡単に call 系の命令に翻訳できる。一方、表2の (b) で左辺の変数が関数項に束縛されている場合、翻訳時にはどのような関数項が束縛されるかはわからないので、変数に束縛されている関数を呼び出す命令を用意しなければならない。この目的のために命令 push_args と call_function を追加する。push_args は、A レジスタにセットされた関数項から引数を取り出しスタックに積む。call_function は、A レジスタにセットされた関数項から関数名を取り出し、対応する手続きを呼ぶ。

効率上、同じ関数項は一度だけしか書換えが起こらないようにすることが望ましい。このために、関数項にキャッシュフィールドと呼ぶ領域を確保し、書換えによって得られた値をここに保存する。この目的のために bind 系命令を追加する。この命令は push 系命令のように等式の右辺の値をスタックに積むと同時に、Aレジスタにセットされた関数項のキャッシュフィールドに等式の右辺の値をセットする。

書き換えによって得られた値をキャッシュフィールドに保存するので、jmp_if_function 命令の仕様は以下のようになる。

- 引数をデレファレンスした結果が関数項であるとき、キャッシュフィールドに値が保存されていないときだけ指定したラベルに分岐する。

4.3 翻訳例

Ev の基本式 $f(X,Y)=Y \Leftarrow X=[Z \mid W]$. の中の等式 $X=[Z \mid W]$ の翻訳結果を例として示す。ここではWAMの記法にならって引数 X,Y を A0,A1 で、環境に割り当てられる変数 Z,W を Y0,Y1 で示した。

```
        jmp_if_function  A0, $L1
        get_list
        unify_variable   Y0
        unify_variable   Y1
        jmp              $L2
$L1 :   push_args
        bind_list
        unify_variable   Y0
        unify_variable   Y1
        call_function
$L2 :
```

jmp_if_function 命令により、A0 をデレファレンスした結果がAレジスタにセットされる。この値が関数項でないときには、get_list 命令以下の命令列が実行される。このときは、Aレジスタの内容とリストとの単一化が試みられる。また、get_list 命令は、Aレジスタを単一化の対象とするため引数が省略されている。

A0 をデレファレンスした結果が関数項であるときは、ラベル $L1 以下の命令列が実行される。このときは、Aレジスタにセットされた関数項を書き換えるために、関数項に対応する手続きが呼び出される。

2で示した基本式の性質より、等式の右辺は必ず浅いデータ項なので、右辺がリストや構造体のときには、その要素には新しい変数しか現れない。このため、この例のように等式の翻訳が Prolog の場合と比べて単純なものになる。

5 おわりに

本稿では、言語 Ev の等式の翻訳方法について考察し、必要な命令セットが WAM を拡張したもので良いことを示した。ここで示した翻訳に基づく翻訳系が既に開発され、Ev の処理系に実装されている [1]。

参考文献

[1] 鈴木太朗、中川康二、井田哲雄：遅延ナローイング計算系に基づく言語 Ev とその処理系，情報処理学会第44回全国大会予稿 5F-7,1992

[2] 中村敦司、鈴木太朗、中川康二、井田哲雄：遅延ナローイング抽象機械のアーキテクチャ，情報処理学会第44回全国大会予稿 6G-1,1992

[3] Hassan Aït-Kaci ： Warren's Abstract Machine, MIT Press,1991