7F−5

関数・論理型言語のためのナローイング計算系

鈴木太朗　井田哲雄
筑波大学

1 はじめに

等式を用いたプログラミングにより、関数型言語と論理型言語を自然に融合した言語(以後、関数・論理型言語と呼ぶ)を定義できる。本稿では、strict equal と呼ばれる特別な等号を持つ関数・論理型言語のためのナローイング計算系について述べる。

2 関数・論理型言語における等号

等式をプログラムとみるとき、等式は左辺から右辺への書換え規則とみなされる。すなわち、等式の集合を与えたとき、それを関数定義の集合とみなすことができる。たとえば、$f(x) = b(x)$ のような等式は関数 f の定義であるとみなせる。関数記号ではじまる項を関数項、それ以外の項を構成子項と呼ぶ。また、部分項に関数項を持たない項をデータ項、部分項に関数項および変数を持たない項を閉データ項と呼ぶ。

関数・論理型言語を設計する際、等式の両辺が「等しい」という意味をどのように定めるかが重要になる。等式 $g(x) = g(x)$ の下で、ゴール $g(a) = y$ を解く方法として、単一化代入 $\{g(a)/y\}$ により、$g(a) = g(a)$ を得るというものが考えられる。しかし、等式を関数定義とみなすと $g(a)$ は関数項なので、このような結果はプログラムの実行結果として不十分である。プログラムの解としては少なくとも関数項以外の項を返すことが望ましい。したがって、両辺が等しい関数項であるときに両辺が「等しい」という解釈は、プログラミング言語としては不十分である。

等式の両辺が「等しい」ということを関数・論理型言語の設計の観点から考えたとき、単一化代入と、与えられた等式の集合から定義される関数とによって、等式の両辺が等しい閉データ項に書換え可能であるときのみ両辺が「等しい」とするのは一つの自然な解釈である。このような解釈の下では、通常の等号 $(=)$ よりも *strict equal* と呼ばれる特別な等号 $(\equiv^?)$ を用いる方が望ましい。*strict equality* $t \equiv^? s$ は $f_{\equiv}(t,s) = \text{true}$ の略記である。$f_{\equiv}$ は関数記号で、与えられた等式の集合に関して閉データ項である全ての d について、$f_{\equiv}(d,d) \to \text{true}$ で表わされる書換え規則の集合により定義される。この集合を $R_{\equiv}$ と呼ぶ。次節では、$R_{\equiv}$ を組込の書換え規則の集合として持つ計算系 NCS について述べる。

3 ナローイング計算系 NCS

本節では、strict equality の書換え規則の集合 $R_{\equiv}$ を組込として持つナローイング計算系 NCS を定義する。NCS は言語 $\mathcal{L}$ と推論規則の集合から成る。

[0]Narrowing Calculus for Functional-Logic Programming Languages,
Taro Suzuki and Tetsuo Ida,
University of Tsukuba

$[\Phi_1]$

$$\begin{aligned}&\Phi_1[\![f(\ldots,d,\ldots) = s \Leftarrow E]\!]\\ =\ &\Phi_1[\![f(\ldots,x,\ldots) = s \Leftarrow x = d, E]\!]\end{aligned}$$

d は変数ではないデータ項、x は新たに導入された変数。

$[\Phi_2]$

$$\begin{aligned}&A \Leftarrow \Phi_2[\![\ldots, s = c(\ldots,d,\ldots),\ldots]\!]\\ =\ &A \Leftarrow \Phi_2[\![\ldots, s = c(\ldots,x,\ldots), x = d,\ldots]\!]\end{aligned}$$

d は変数ではないデータ項、x は新たに導入された変数。

図1. 基本式への変換規則 $\Phi = \Phi_2 \circ \Phi_1$

3.1 言語 $\mathcal{L}$

言語 $\mathcal{L}$ は、組込の書換え規則の集合 $R_{\equiv}$ とプログラム及びゴール式から成る。プログラム $\mathcal{P}$ は以下のような条件付き等式の集合である。

$$f(d_1,\ldots,d_n) = s \Leftarrow t_1 \equiv^? s_1, \cdots, t_m \equiv^? s_m$$
ただし、$m,n \geq 0$

$\mathcal{P}$ には、以下の制限がある。

C1. $d_1,\ldots,d_n$ はデータ項

C2. P 中の全ての条件付き等式 $l = r \Leftarrow F$ について、l は線型

C3. P 中のどの2つの条件付き等式の間にも重なりがない。

$\mathcal{P}$ 中の条件付き等式は、基本式に変換される。基本式が NCS の推論の対象となる。基本式とは、$f(x_1,\cdots,x_n) = s \Leftarrow t_1 = d_1, t_m = d_m$ のように、f の引数がすべて変数であり、かつ $d_1,\cdots,d_m$ がすべて $c(y_1,\ldots,y_l)$ のような形式 $(l \geq 0)$ のデータ項であるような条件付き等式である。基本式への変換は、図1に示す変換規則 Φ により行なわれる。プログラム $\mathcal{P}$ 中の条件付き等式を基本式に変換して得られるプログラムを basic($\mathcal{P}$) で表す。

3.2 NCS の推論規則

NCS の推論規則を以下に示す。

[v] 変数の除去

$$\frac{\Leftarrow x = d, E}{\Leftarrow \theta E} \quad \theta = \{d/x\}$$

[on] 最外ナローイング

$$\frac{\Leftarrow t = d, E}{\Leftarrow \theta(F, t[u \leftarrow r] = d, E)}$$

tは関数項
$l = r \Leftarrow F \in \mathrm{basic}(\mathcal{P})$
t/uは最左最外ナローイング可簡約項
$\theta l \equiv \theta(t/u)$

[u] 構成子項の単一化

$$\frac{\Leftarrow c(t_1, \ldots, t_n) = c(x_1, \ldots, x_n), E}{\Leftarrow \theta E} \quad \theta = \{t_1/x_1, \cdots, t_n/x_n\}$$

図2：NCS の推論規則

図2中で、E, F は等式の列、c は構成子記号である。[on] は等式 $t = d$ の左辺 t に含まれる関数項を書き換えるための規則である。t は $\mathcal{P}$ で定義されている関数項か、項 $f_{\equiv}(p, q)$ である。[on] は常に最左最外ナローイング可簡約項を書き換えるので、後者の場合 p, q がデータ項 p', q' になるまで書き換えてから $\mathcal{R}_{\equiv^?}$ 中の書換え規則を用いて $f_{\equiv}(p', q')$ を書き換える。

NCS での実行は、与えられたゴールに対して推論規則を適用していき、得られたゴールに対して再び推論規則を適用するという過程を繰り返すことである。

NCS では、ゴール中の最も左にある等式が推論規則の適用の対象となる。また、等式の左辺の項のタイプにより適用すべき推論規則は一意に定まる。これらは、効率の良い処理系実現にとって都合が良い。また、基本式への変換と NCS の推論規則により、$\mathcal{P}$ で定義された関数項の書換えの際「必要による呼び出し」(call by need) [2] に相当することが実現できる。このため、空ゴール (□) を求めるのに必要のない項は書き換えられないことが保証される。この点については、次節で例を示す。

4 NCS の実行例

次のようなプログラム $\mathcal{P}$ とゴール $\mathcal{G}$ が与えられているとする。

$$\mathcal{P} = \begin{cases} f(c(z), y) = a \Leftarrow y \equiv^? c(a) \\ g(x) = c(g(x)) \end{cases}$$

$$\mathcal{G} = k(f(g(b), c(a))) \equiv^? k(a)$$

基本式の集合である $\mathrm{basic}(\mathcal{P})$ は以下のようになる。

$$\mathrm{basic}(\mathcal{P}) = \begin{cases} f(x, y) = a \Leftarrow x = c(z), y \equiv^? c(a) \\ g(x) = c(g(x)) \end{cases}$$

NCS による実行過程を図3に示す。図3中の [on]+[u] は、$c(a) \equiv^? z$ のように $\equiv^?$ の両辺がデータ項であるときに、$R_{\equiv}$ 中の書換え規則を用いて [on] を適用して得られる等式 true = true にただちに [u] を適用することを表わしている。図3では、$g(b)$ は一度書き換えられて $c(g(b))$ になった時点で、それ以上の書換えは必要ないため行なわれない。このように、NCS によって必要による呼び出しを実現できることがわかる。

$$\begin{array}{l} \underline{k(f(g(b), c(a)))} \equiv^? k(a) \\ \quad \Downarrow [on] \\ \underline{g(b) = c(w)}, c(a) \equiv^? c(a), k(a) \equiv^? k(a) \\ \quad \Downarrow [on] \\ \underline{c(g(b)) = c(w)}, c(a) \equiv^? c(a), k(a) \equiv^? k(a) \\ \quad \Downarrow [u] \\ \underline{c(a) \equiv^? c(a)}, k(a) \equiv^? k(a) \\ \quad \Downarrow [on] + [u] \\ \underline{k(a) \equiv^? k(a)} \\ \quad \Downarrow [on] + [u] \\ \square \end{array}$$

図3：NCS による実行過程

5 K-LEAF との比較

関数・論理型言語の設計に対して、本稿と同様のアプローチをとったものとして K-LEAF[1] がある。しかし、その計算機構は本稿のものと K-LEAF とでは全く異る。

K-LEAF の場合、プログラムを関数項の入れ子がない平坦 (flat) なプログラムに変換し、SLD 反駁手続きを用いて実行する。この方法は Prolog の計算機構としてよく知られた SLD 反駁を用いる点や、最内戦略の基底ナローイングが通常の Prolog と同じ最左の等式から順に選択していく単純な戦略の SLD 反駁で実現できる点などで優れている。しかし、この方法で関数項の必要による呼び出しのような戦略を扱おうとすると、等式への SLD 反駁の適用の一時的な中断、再開など等式の選択の順序を実行時に決める複雑な戦略を用いなければならない。

一方、我々の計算系では、前節で示したように NCS の推論規則では常に左側の等式が選ばれるので、等式の選択に関する戦略がひじょうに単純になる。

6 おわりに

本発表では、strict equal という特別な等号を持つ関数・論理型言語のための単純で効率の良い計算系について述べた。現在、この計算系に基づく抽象機械を設計している。

参考文献

[1] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel-LEAF: a logic plus functional language. *J. Compt. Syst. Sci.*, 42(2):139–185, 1991.

[2] G. Huet and J. J. Lévy. Computations in orthogonal rewriting systems,I. In J. L. Lassez and G. Plotkin, editors, *Computational Logic*, chapter 11, pages 395–414. The MIT Press, 1991.