リフレクティブな書換え計算モデル RRC

河口信夫　　坂部俊樹　　稲垣康善

(名古屋大学 工学部)

1 はじめに

自分自身の計算の状態をモデルとして持ち, その参照・変更が状態・挙動に反映するシステムをリフレクティブ (自己反映的) なシステムと呼び, このような計算をリフレクション [2] と呼ぶ. リフレクションはシステムの動的な変更, デバッグ, 検証などに有用で, 近年研究が盛んに行なわれている.

リフレクションの機能を書換え形の計算モデルに導入することにより, システムの動的な変更やプログラムの等価変換, プログラムの自動検証, デバッグのサポートなどが可能となる. 本研究ではリフレクティブな書換え計算モデル (Reflective Rewriting Calculus: RRC) を提案し, その有効性を示す.

2 リフレクティブな書換え計算モデル

リフレクティブなシステム S には, 内部にその自己表現 R_S が存在する. S が状態を変化させるにしたがって, R_S も変化し, 逆に R_S が S により変更されると, S もその変更に対応して変化する. このような関係を S と R_S の causal connection と呼ぶ. R_S が S のすべての部分を表現しているとすると, リフレクティブな計算を行なう部分にまでリフレクションをすることになり reflective overlap[1] という問題が起こる. そこでシステムはメタな部分 (メタシステム) とそうでない部分 (サブシステム) に区分けされる.

RRC では, メタシステムとサブシステムを区別するために ↑(quote) および ↓(unquote) を導入する. ↑ は項をデータとして, すなわちサブシステムとして扱うためのものであり, ↓ はサブシステムをメタシステムに戻す働きをする. RRC の形式的な定義は以下のとおりである.

【定義 2.1】記号: RRC の記号は, 変数記号, 関数記号 (ただし, 特殊な関数記号の集合 $F_{sys} = \{Sys, Rule, cons, nil, stop, run, step\}$ を含む), 2 つのメタ記号 (↑, ↓), 区切り記号 ",","(",")" からなる.

各々の関数記号には非負整数 n が割り当てられ, これを関数のアリティと呼ぶ. アリティ n の関数記号 f は f/n と表記する. $Sys/3$, $Rule/2$, $cons/2$, $nil/0$ とする. また $stop/0$, $run/0$, $step/0$ であり, これらをアクションと呼ぶ. □

【定義 2.2】メタ定数記号: 変数記号, 関数記号, メタ定数記号に ↑ がついたものをメタ定数記号という. □

【定義 2.3】項 (term) : 項は次のように定義される.

1. 変数記号, メタ定数記号は項である.
2. $t_1, \ldots, t_n$ が項, 関数記号 f がアリティ n である時, $f(t_1, \ldots, t_n)$ は項である. □

【定義 2.4】quote: 項 T に対して ↑T は次のように定義される. ↑T は quote T と呼ぶ.

1. T が変数記号, メタ定数記号の時, ↑T はそれ自身メタ定数記号である.
2. T が $f(t_1, \ldots, t_n)$ で, $t_1, \ldots, t_n$ が項, f がアリティ n の関数記号であるとき, $\uparrow f(t_1, \ldots, t_n) = [\uparrow f, \uparrow t_1, \ldots, \uparrow t_n]$ である. ただし, $[t_1, \ldots, t_n]$ は次で定義するようにリストを表す.

また, $\downarrow\uparrow T = T$, すなわち, ↓ は ↑ の逆を表し unquote と呼ぶ. □

【定義 2.5】特別な項: リスト, ルール, システムは次のように特別に定義される.

1. リストは関数記号 $cons$, nil により次のように定義される項である. nil はリストで空リストを表す. t を項, l をリストとすると $cons(t, l)$ はリストである. 項 $t_1, \ldots, t_n$ のリストを省略して, $[t_1, \ldots, t_n]$ と表す.
2. ルールは関数記号 $Rule$ と, 左辺の項 $t1$, 右辺の項 $t2$ によって $Rule(t1, t2)$ と定義される. 右辺には左辺に存在しない変数記号を含むことは許されない. R_i をルールとしたとき, ルールのリスト $[R_1, \ldots, R_n]$ をルールリスト $\mathcal{R}$ と呼び, また, $[\uparrow R_1, \ldots, \uparrow R_n]$ を $\uparrow\mathcal{R}$ と表す.
3. システムは関数記号 Sys と quote されたルールリスト $\uparrow\mathcal{R}$, quote された項 ↑t, アクション act により $Sys(\uparrow\mathcal{R}, \uparrow t, act)$ と定義される. □

【定義 2.6】書換え: 項 t をルール R によって書換えるとは, t の部分項 u と R の左辺が適当な代入 σ によって等しくなる時, u を R の右辺に代入 σ を施したもので置き換えることにより t から t' を得ることである.

ルールリスト $\mathcal{R}$ による書換えとは, リスト中の任意のルール R_i を用いて書換えを行なうことである. □

【定義 2.7】遷移: 特別な項 $Sys(\uparrow\mathcal{R}, \uparrow t, act)$ の遷移は次のように定義される. アクション act が stop の時は遷移しない. その他の時は, $\uparrow\mathcal{R}$, ↑t はそれぞれ $unquote$ され, $\mathcal{R}$ により t から t' への書換えが 1 回行なわれる. もし act が step ならば act' は stop になる. act が run で t' がそれ以上書換えできず, 遷移もしないならば, act' は stop になる. それ以外では act' は run のままである. 書換えられた t' は $quote$ され, $Sys(\uparrow\mathcal{R}, \uparrow t', act')$ が遷移の結果となる. □

3 RRC のプログラム例 (システムの生成)

$\uparrow Rule(x, y)$ を $x \rightarrow y$ で表す.

$$Sys([cons(Sys(x, \uparrow fork, a), y) \rightarrow cons(Sys(x, \uparrow parent, a), cons(Sys(x, \uparrow child, a), y))], \uparrow cons(Sys([child \rightarrow fork], \uparrow fork, run), nil), run)$$

このプログラムはシステムを無限に生成しつづける. $fork$ を適当に変更すれば有限停止のプログラムに簡単に修正できる.

4 まとめ

リフレクションの機能を導入した書換え形の計算モデルとして RRC を提案した. アクションとメタ記号を採用することによって, システムの状態の参照・変更を柔軟に行なうことが可能となり, リフレクティブな計算の簡単な記述が可能となった.

参考文献

[1] Maes,P.: Issues in Computational Reflection ,Maes, P. and Nardi,D. Ed.:*Meta-Level Architectures and Reflection* , North-Holland, pp.21-35(1988).

[2] Smith,B.C.:Reflection and semantics in lisp ,In *Proc. 11th ACM Symposium on Priciples of Programming Languages* , pp.23-35(1984).

[3] 菅野, 田中: メタ推論とリフレクション, 情報処理, Vol.30, No.6, pp.694-705(1989).