

プログラム言語論 第4週 補足資料

亀山幸義, 2009/05/11

1 miniML の構文を定義する。

変数 x と定数 c は、miniC と同じ。

$$\begin{aligned} \text{式 } e, f ::= & x \mid c \mid e + f \mid e = f \mid e > f \mid \text{print } e \\ & \mid \text{if } e \text{ then } e \text{ else } e \mid \text{fun } x \rightarrow e \mid e \text{ } f \\ & \mid \text{let } x = e \text{ in } f \mid \text{let rec } x = e \text{ in } f \end{aligned}$$

ラムダ式 (1 引数の関数):

- $f(x) = e$ となる関数 f のことを $\lambda x.e$ と書く。
- $\lambda x.e$ では、 x は局所変数 (外からは見えない変数)。
- $(\lambda x.e)(x) = e$ が成立する。
- ML の一種である OCaml 言語では、 $\lambda x.e$ を `fun x -> e` と書く。

関数適用 (関数呼出し):

- $e_1 \ e_2$ と、ただ並べて書く。(括弧をつけてもよい。)
- e_1 の計算結果が、関数 `fun x ->  $e_0$` になったら、その関数を、 e_2 という引数で呼び出し、その結果を返す。
- $(\text{fun } x \rightarrow x + 1)(2 + 3)$ は、6 になる。

let 式 (`let  $x = e$  in  $e'$`)

- $x = e$ という (局所的な) 環境のもとで e' を計算する。
- $(\text{fun } x \rightarrow e')e$ と同じ計算になる。

let rec 式 (`let rec  $f = e$  in  $e'$`)

- e' の中で f を使ってよい (再帰呼び出し)。

2 miniML の意味を定義する。

「式」しかない (「文」はない) のと、「状態変化」がないので、miniC の意味論よりだいぶ簡単になる。

2.1 値

まず、計算結果を意味する「値」(あたひ、value) を以下のように定義する。

$$\text{値 } v ::= x \mid c \mid (\text{fun } x \rightarrow e, \sigma)$$

ここで、 $(\text{fun } x \rightarrow e, \sigma)$ というのが不思議な値である。関数型言語では、計算結果が「関数」になることがあり、それを後で使えるようにするため、関数本体と、その関数を計算したときの環境とをセットにしたものをクロージャ (closure、閉包) と呼ぶ。クロージャがなぜ必要か、単純に、関数 `fun x ->  $e$` を計算結果としては駄目なのか、について、講義を参照のこと。

2.2 式の意味

環境 σ のもとで、式 e を計算した結果を $\sigma(e)$ と書く。これを e に関する場合分けで定義する。

- $e = x$ のとき: $\sigma(e)$ は、 σ における x の値である。
- $e = c$ のとき: $\sigma(e) = c$ である。
- $e = e_1 + e_2$ のとき: $\sigma(e) = \sigma(e_1) + \sigma(e_2)$ である。
- $e = \text{if } e_1 \text{ then } e_2 \text{ else } e_3$ のとき: $\sigma(e_1) = \text{true}$ のとき、 $\sigma(e) = \sigma(e_2)$ で、 $\sigma(e_1) = \text{false}$ のとき、 $\sigma(e) = \sigma(e_3)$ である。それ以外の場合、 $\sigma(e)$ は値を持たない。
- $e = (\text{let } x = e_1 \text{ in } e_2)$ のとき、 $e = (\text{fun } x \rightarrow e_2)e_1$ と見なして計算する。(let rec は、少し面倒)
- $e = \text{fun } x \rightarrow e_1$ のとき: $\sigma(e) = (\text{fun } x \rightarrow e_1, \sigma)$ である。
- $e = e_1 e_2$ のとき:
 - $\sigma(e_1) = (\text{fun } x \rightarrow e_3, \sigma_3)$ かつ、 $\sigma(e_2) = v$ となるとき: $\sigma_4 = ((x = v) \oplus \sigma_3)$ とおいた上で、 $\sigma(e) = \sigma_4(e_3)$ である。
 - 上記のようにならないとき、 $\sigma(e)$ は値を持たない。

以上。(print と let-rec の意味は省略した)

3 miniML のインタープリタ

インタープリタは、意味論をそのまま反映したプログラムである。miniC のインタープリタは、意味論が複雑なことに対応して結構複雑であった。(文や式の定義を相互再帰的に呼んでいた。) 一方、miniML の方は、意味論が非常にすっきりしているので、対応してインタープリタもだいぶ簡単である。ほとんど上記の定義をそのまま書くだけ、といった感じになる。詳細は、演習の際に学習してほしい。

4 miniML の拡張

演習で使う miniML は、ここまで述べたものに、さらに「直積」を追加したものである。

式の構文の追加:

$$\text{式 } e, f ::= \dots \mid (e, f) \mid \text{fst } e \mid \text{snd } e$$

直感的には、 (e, f) は e と f の計算結果 v_1, v_2 を「対 (つい、ペア)」にしたデータのことである。(要素数 2 の配列、あるいは、C 言語の struct 型で 2 要素の構造体を作ったときのデータと似ている。)

$\text{fst } e$ と $\text{snd } e$ は、それぞれ、対の第 1、第 2 要素である。

式の意味の追加:

- $e = (e_1, e_2)$ のとき: $\sigma(e) = (\sigma(e_1), \sigma(e_2))$ である。
- $e = \text{fst } e_1$ のとき: $\sigma(e_1) = (v_2, v_3)$ ならば、 $\sigma(e) = v_2$ であり、それ以外の場合、値を持たない。
- $e = \text{snd } e_1$ のとき: $\sigma(e_1) = (v_2, v_3)$ ならば、 $\sigma(e) = v_3$ であり、それ以外の場合、値を持たない。