プログラム言語論 補足資料 (MiniML)
亀山

1 ラムダ式

- $f(x) = e$ 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 $\lambda x.e$ と書く。

2 MiniML の構文の定義

OCaml のサブセットとして、MiniML を定義する。
変数 x と定数 c の構文は、MiniC と同じ (定数は、整数定数と真偽値定数 (true, false))。

$$
\begin{aligned}
\text{式}\quad e ::= {} & x \mid c \mid e + e' \mid e * e' \mid e = e' \mid e > e' \\
& \mid \texttt{if}\ e\ \texttt{then}\ e'\ \texttt{else}\ e'' \mid \lambda x.e \mid e\ e' \\
& \mid \texttt{let}\ x = e\ \texttt{in}\ e' \mid \texttt{let rec}\ x\ x' = e\ \texttt{in}\ e' \\
& \mid (e, e') \mid \texttt{fst}(\ )e \mid \texttt{snd}(\ )e \\
& \mid (e) \mid p\ e
\end{aligned}
$$

(p は、プリミティブな関数のことで、演習の際に、適宜用意する。)

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 (を計算した結果) が関数となり、それを引数 e_2 に適用 (apply) する。
- $(\lambda x.x + 1)(2 * 3)$ は、7 になる。

let 式 ($\texttt{let}\ x = e\ \texttt{in}\ e'$)

- $x = e$ という局所的な環境のもとで e' を計算する。
- $(\lambda x.e')\ e$ と同じ計算になる。
- 例: $\texttt{let}\ x = 1 + 2\ \texttt{in}\ x * 2 + 1 = 7$)

let rec 式 ($\texttt{let rec}\ f\ x = e\ \texttt{in}\ e'$)

- e' の中で f を使ってよい (再帰呼び出し)。

注意すべき点: 関数は 1 引数のものしかない。変数に値を代入することはできない。C 言語の for 文や while 文はなく、関数の再帰呼び出しを使ってループ構造を実現する。
プログラム例:

```
(let x = 1 in x + x) + 5
     ==> 7
fun x -> x + 1
     ==> (関数が返ってくる)
let f = (fun x -> x + 1) in f (f 3)
     ==> 5
(fun f -> f (f 3)) (fun x -> x + 1)
     ==> 5
(fun f -> fun x -> f (f x)) (fun y -> y + 1) 3
     ==> 5
let x=5 in let f=(fun y->x+y) in let x=10 in (f 20)
     ==> (これが何になるか考えてほしい)
let x=5 in let f=(fun y->x+y) in let x=10 in (f 20)
     ==> (これが何になるか考えてほしい)
```

3 「対」のデータ

式の構文の最後の行：

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

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。)

`fst`$(\ )e$ と `snd`$(\ )e$ は、それぞれ、対の第 1、第 2 要素である。

- $(1 + 2, 3 * 4) = (3, 12)$
- $\texttt{fst}(()1 + 2, 3 * 4) = 3$

※ OCaml では、(厳密にいえば) e, f と括弧なしで書いても「対」と見なされる。MiniML では必ず括弧をつけることにする。

4 例題

・let 式を使った簡単な例:

```
let x = 1 in
  let y = 2 in
    x + y

let f = (fun x -> x + 1) in
  f (f 3)

(let x = 1 in
   x + x)
+ 5
```

```
(fun x -> if x = 1 then 2 else 3) 1

(fun x -> if x = 1 then 2 else 3) 10
```

・ブロック構造，変数のスコープ

```
let x=5 in
  (let f=(fun y->x+y) in
    let x=10 in
     (f 20) + x * 2)
  + x * 3

let rec f x =
     if x=0 then 1
     else x * f (x-1)) in
  (f 10)
```

・関数をデータとして扱う

```
let f = (fun x -> fun y -> x + y) in
  f 3 5

let f = (fun x -> fun y -> x + y) in
  (f 3) 5

let f = (fun x -> fun y -> x + y) in
  let g = f 3 in
    (g 5) + (g 7)

(fun f -> f (f 3)) (fun x -> x + 1)

(fun f -> fun x -> f (f x)) (fun y -> y + 1) 3
```

・値呼び，名前呼び

```
let f = fun x -> (print x; 2) in
  (fun y -> y + y + y) (f 1)

let f = fun x -> (print x; 2) in
  (fun y -> (y 1) + (y 1) + (y 1)) f

let f = fun x -> (print 1; 5) in
  let g = fun x -> (print 2; 7) in
    f g
```

・対 (ペア) を表すデータ型

```
(1,2)

fst (1,true)
```

```
fst (snd (1,(true,3)))

(fun x -> fun y -> ((x,y),(y,x))) 10 true

(fun x -> (fst x) ((snd x) 10))
   ((fun z -> z * 3), (fun y -> y + 2))
```

5　処理系の起動、参考書

OCaml の起動は、単に ocaml というコマンドを打てばよい。(MiniC 演習のときに、処理系の抜け方も含めて、説明済み。)

MiniML 処理系をもし動かしたければ、`~kam/miniml/miniml` を起動するとよい。

参考: OCaml の使い方について: インターネット上に解説テキストがいくつかあるので検索してみるとよい。本家 (フランス INRIA 研究所) の解説は、`http://caml.inria.fr/pub/docs/manual-ocaml/` である。また、日本語での教科書も 3 冊 (それぞれの著者は、京都大学五十嵐淳氏、お茶の水女子大学浅井健一氏、OCaml-Nagoya グループ) 出版されている。