プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 2

授業の全体像

先週のメッセージの 1 つ

- プログラムの意味を、今年は OCaml (または F#) で書く

今日の 3 限: 講義

- 教科書 1 章の積み残し: 式 (expression)
- 教科書 2 章前半: スコープ, 束縛、静的束縛、閉じた式、自由変数
- (来週) 教科書 2 章後半: スタック機械、コンパイル、 PostScript
- (将来) 3 章は飛ばして、4 章へ行く

今日の 4 限: 演習

- まずは OCaml (または F#) に慣れることが必要
- 教科書 pp.245-255 の範囲
- 今日のレポート: 初めての人向けの課題と、「ソフトウェア技法」等で OCaml を勉強したことのある人向けの課題を両方出題して、どちらかを解答 (manaba システムから提出)

式と文

Expression (式)

```
17
3-4
7.9 + 10
```

Statement (文)

```
printf("The answer is %d\n", 17);
x = y * 2 + 1;
```

プログラム言語によっては、どちらかしかないこともある。

- C や Java では両方ある。
- OCaml では、print 文に相当するものも式である (unit 型を返す式)。

簡単な式 (算術式) の構文

式 e の構文 in BNF (i は整数定数、p はプリミティブ演算子):

$$e ::= i \mid p(e,e) \mid (e) \quad \text{具体構文}$$
$$e ::= \texttt{Int}(i) \mid \texttt{Prim}(p,e,e) \quad \text{抽象構文}$$

抽象構文を、OCaml のデータ型を使って、以下のように表現する。

```
type expr =
  | CstI of int                    (* CstI(i) *)
  | Prim of string * expr * expr   (* Prim(p,e1,e2) *)
```

ただし、p は文字列として表現する。

例 1. `CstI(10)`
例 2. `Prim("+",CstI(2),CstI(-3))`
例 3. `Prim("+",CstI(20),Prim("*",CstI(-3),CstI(5)))`

OCaml の型

型とは何か？

Short answer → 「計算論理学」で学んでください。

OCaml の型

とても大事: 型が整合しないプログラムは一切受け付けてくれない。

基本型: int, bool, float, string など。

型構成子: 「○ list」、「○ array」など。

ユーザ定義の型: 後述。

OCaml のユーザ定義の型

2 分木:

```
type binary_tree =
  | Leaf of int
  | Node of binary_tree * binary_tree
```

例 1: `Leaf(13) : binary_tree`
例 2: `Node(Leaf(13),Node(Leaf(5),Leaf(7))) : binary_tree`
BNF に近い。木をあらわしている。

OCaml のユーザ定義の型

ちょっと違う 2 分木:

```
type binary_tree' =
  | L of float
  | N of float * binary_tree' * binary_tree'
```

例 1: `L(13.5) : binary_tree'`
例 2: `N(3.1,L(13.5),N(4.1,L(5.3),L(7.0))) : binary_tree'`
自分で自由自在に「木」のような構造を定義できる。

意味

この小さな言語の意味論を厳密に記述してみよう!

$$[\![\mathtt{Int}(n)]\!] = n$$
$$[\![\mathtt{Plus}(e_1, e_2)]\!] = [\![e_1]\!] \dot{+} [\![e_2]\!]$$

その他の形の e に対しては $[\![e]\!]$ は未定義とする。

```
let rec eval e =
 match e with
  | CstI i    -> i
  | Prim("+", e1, e2) -> (eval e1) + (eval e2)
  | _ -> failwith "unknown expression"
```

たったこれだけ。

OCamlで書いたインタプリタ

```
let rec eval e =     再帰関数 eval の定義を始める、仮引数は e
 match e with        e に関するパターンマッチ
  | CstI i    -> i   e が CstI i の形なら i を返す。
  | Prim("+", e1, e2)    e が Prim(...) 形なら
      -> (eval e1) + (eval e2)
                        e1 の計算結果と e2 の計算結果を足し
                        たものを返す
  | _ -> failwith "unknown expression"
                        e が上記以外の形なら、fail する
```

let rec は再帰関数の定義の始まりに使う key word.
match ... with ... | ... | ... はパターンマッチ。
failwith "..." は、エラーメッセージを出して異常終了。

(参考) 抽象機械による意味論

CK 機械 (C=constrol string or code, K=continuation or stack)

$$
\begin{aligned}
e &\to \mathtt{eval}\langle e \mid \mathtt{init}\rangle \\
\mathtt{eval}\langle \mathtt{Int}(n) \mid K\rangle &\to \mathtt{apply}\langle K \mid n\rangle \\
\mathtt{eval}\langle \mathtt{Plus}(e_1, e_2) \mid K\rangle &\to \mathtt{eval}\langle e_1 \mid \mathit{plus}1(e_2)::K\rangle \\
\mathtt{eval}\langle \mathtt{Times}(e_1, e_2) \mid K\rangle &\to \mathtt{eval}\langle e_1 \mid \mathit{times}1(e_2)::K\rangle \\
\mathtt{apply}\langle \mathit{plus}1(e)::K \mid n\rangle &\to \mathtt{eval}\langle e \mid \mathit{plus}2(n)::K\rangle \\
\mathtt{apply}\langle \mathit{plus}2(n_2)::K \mid n_1\rangle &\to \mathtt{apply}\langle K \mid n\rangle \quad (n_2 \dot{+} n_1 = n) \\
\mathtt{apply}\langle \mathtt{init} \mid n\rangle &\to n \quad (\text{最終結果})
\end{aligned}
$$

式 e が与えられると、上記の状態遷移を行う。もし状態 n に行けば、実行は終了する。

今年はこれはやりません!