2017 年度「プログラミング言語処理系」期末試験問題

2017 年 7 月 24 日

■注意

- 問題は全部で *4* 問である．それぞれの問題について解答用紙を 1 枚ずつ使うこと．
- すべての解答用紙に学籍番号，名前等の情報を記入すること．
- 解答用紙毎に，どの問題についての解答かを明示すること．
- 解答用紙の裏面に答案が続く場合は，表面の最後に「裏面に続く」と大きな字で書くこと．
- 以上の事項を守らない答案は採点しないことがある．

問題は次のページから始まる

問題 1

図 1 は講義で扱った $\mathbf{ML}^4$ インタプリタの実装の一部である．ただし，`Environment` は環境 (environment) を表す抽象データ型を表現するモジュールである．

1. 環境とは何か．環境が保持する情報と，環境がインタプリタにおいてどのように用いられているかを簡潔に説明せよ．
2. 関数 `eval_exp` を空の環境と式 `e` で呼び出したところ，図 1 中下線部 (a) が引かれている場所が実行されたという．このような $\mathbf{ML}^4$ の式 `e` を一つ書け．
3. 図 1 のコードは関数 `eval_exp` の定義中の `LetRecExp` のケースにおいて誤りを含むため，`let rec` 式を正しく扱えない．
   (a) 図 1 のコードで正しく扱えない `let rec` 式を一つ書き，期待される出力と，このコードが返す出力を書け．
   (b) `let rec` 式を正しく扱うためには，図 1 のコードをどのように修正すればよいか説明せよ．

問題 2

図 2 は講義で扱った単相的な型システムの型付け規則である．ただし，e は $\mathbf{ML}^4$ プログラムの式を表すメタ変数，τ は型を表すメタ変数である．

1. 以下の式は空の型環境 $\emptyset$ の下で図 2 の型付け規則で型を持つか．型を持つならば，そのことを示す型判断の導出木を，できないならば「できない」と書け．導出木を書く際には，各ステップで使った規則の名前を明示すること．
   (a) $(\texttt{fun}\ x \to x)\ (\texttt{fun}\ y \to y)$
   (b) $\texttt{let rec}\ f\ x = 1 + (f\ x)\ \texttt{in}\ f\ 0$
2. この型システムでは実行時型エラーが起こらないのに型付け可能ではないようなプログラムが存在する．そのようなプログラムを一つ書け．

問題 3

プログラミング言語の関数定義や関数呼び出し機構を実装する際には，関数内で用いるデータや，関数が終了した後にプログラムの実行を続けるために必要なデータを**関数フレーム**としてまとめ，関数呼出し時，関数定義の先頭，関数からのリターン時に関数フレームに関する処理を行うことが多い．関数フレームは**スタック領域**と呼ばれるメモリ中の領域に確保されることが多い．

上記のように関数フレームを用いるプログラミング言語の実装において，関数呼出し時，関数定義の先頭，関数からのリターン時に行われるべき関数フレームに関する処理をそれぞれ簡潔に説明せよ．（MIPS アセンブリを書くことを求めているのではなく，どのような処理が行われるかを日本語または英語で説明すればよい．）解答にあたっては以下の条件を考慮せよ．

- スタック領域中で使用されているメモリ領域のうち最小のアドレスを**スタックポインタ**と呼ばれるレジス

タ `$sp` が保持する．

- ターゲットとなるアーキテクチャには関数呼び出し命令 `jal` があり，ラベル l を引数としてこの命令を実行すると，関数から戻って来た直後に実行されるべき命令のアドレスがレジスタ`$ra` にセットされる．
- ターゲットとなるアーキテクチャには命令 `jr` がある．レジスタ r を引数としてこの命令を実行すると，r に書かれたアドレスの命令にジャンプする．
- ソース言語は一引数の関数のみが定義できるものとする．関数呼び出し規約によれば，関数呼出し時にはレジスタ`$a0` に実引数をセットして関数を呼び出し，関数から返る際には返り値をレジスタ`$v0` にセットしなければならない．
- 関数内で使用するローカル変数の値はすべてスタック上に保持されるものとし，関数フレームにはこのための領域が最低限含まれているものとする．その他の必要なデータは適宜含めるようにせよ．
- 講義資料中の該当箇所を再現することは求めていない．上記の条件を満たし，関数呼び出し機構の実装が可能であると採点者が認めれば，講義資料と合致していなくても点を与える．

問題 4

以下の文章を読んで，後の問に答えよ．

以下では集合 S の要素の長さ 0 以上の列の集合を S^* と書く．**文脈自由文法** G は四つ組 (Σ, N, R, S) で表される文法である．ここで

- Σ は**終端記号**の集合，
- N は**非終端記号**の集合，
- R は $A \to \alpha$ の形をした**生成規則**の集合（ただし $A \in N$ かつ $\alpha \in (\Sigma \cup N)^*$）
- $S \in N$ は開始記号

である．**文法 G が生成する言語** $L(G)$ とは R 中の生成規則を S に有限回適用して得られる Σ^* の要素全体の集合である．$\alpha, \beta \in (\Sigma \cup N)^*$ と $(X \to \beta) \in R$ について，α 中の非終端記号 X の出現を一個 β で置き換えて文字列 γ が得られるとき，$\alpha \to \gamma$ と書く．また，$\alpha \to \alpha_1 \to \cdots \to \gamma$ という 0 回以上の書き換えの系列があるときに，$\alpha \to^* \gamma$ と書く．

講義においては LL(1) 構文解析というトップダウン構文解析アルゴリズムを扱った．この構文解析アルゴリズムにおいては，与えられた文脈自由文法 (Σ, N, R, S) について，以下で定義される Nulls, FIRST, FOLLOW の 3 種類の関数の値をあらかじめ計算する．

$$
\begin{aligned}
\text{Nulls} &= \{X \in N \mid X \to^* \epsilon\} \\
\text{FIRST}(X) &= \{a \in \Sigma \mid \boxed{\text{(a)}}\} \\
\text{FOLLOW}(X) &= \{a \in \Sigma \mid S \to^* \alpha_1 X a \alpha_2 \wedge \alpha_1, \alpha_2 \in (\Sigma \cup N)^*\}
\end{aligned}
$$

ただし，ϵ は空列である．列の連結は列を互いに連接するように並べることで表現されており，例えば上記定義中の $a\alpha$ は長さ 1 の列 a と列 α を連結して得られる列である．

直観的に言えば，$\text{FIRST}(X)$ は非終端記号 X を規則 R に従って 0 回以上書き換えたときに一文字目に現れうる終端記号の集合，$\text{FOLLOW}(X)$ は開始記号 S を規則 R に従って 0 回以上書き換えて得られる $(\Sigma \cup N)^*$ の要

素で，(b) の集合である．例えば，開始記号が S の以下の文法

$$\begin{array}{rcl} \Sigma & = & \{\$, +, -, *, (,), \mathbf{ID}\} \\ N & = & \{S, E, G, T, H, F\} \\ R & = & \left\{ \begin{array}{ll} S \to E\ \$ & E \to T\ G \quad G \to \epsilon \mid +E \mid -E \\ T \to F\ H & H \to \epsilon \mid *T \quad F \to \mathbf{ID} \mid (E) \end{array} \right\} \end{array}$$

において，Nulls は (c)，FIRST(S) は (d)，FOLLOW(T) は (e) である．

文法 $G = (\Sigma, N, R, S)$ が与えられ，Nulls が正しく計算できたとしよう．このときに，各非終端記号 $X \in N$ について FIRST を求めるアルゴリズムは以下の通りである．

Algorithm 1 各非終端記号の FIRST を求めるアルゴリズム

```
1: procedure CalcFIRST(Nulls, N, R)                ▷ fst[X] に FIRST(X) の値を計算
2:     for X ∈ N do
3:         fst[X] := ∅
4:     end for
5:     repeat
6:         for X → Y_1 ... Y_n ∈ R do
7:             for i = 1 to n do
8:                 if Y_i ∈ Σ then
9:                     fst[X] := fst[X] ∪ {Y_i}
10:                else
11:                    fst[X] := fst[X] ∪ fst[Y_i]
12:                end if
13:                if Y_i ∉ Nulls then break
14:                end if
15:            end for
16:        end for
17:    until fst が変化しなくなるまで
18: end procedure
```

1. 文中の空欄を正しく埋めよ．答えのみ書け．
2. Algorithm 1 において，13–14 行目のコードがない場合に FIRST を正しく求めることができなくなる文法を一つ与え，なぜその文法で FIRST を正しく求めることができないか説明せよ．
3. Algorithm 1 は任意の文法について停止する．その理由を説明せよ．

```
type id = string

type exp =
  | ...
  | Var of id (* 変数式 *)
  (* 条件分岐式. If(e,e1,e2) で式 if e then e1 else e2 を表す. *)
  | IfExp of exp * exp * exp
  (* 匿名関数を生成する式. FunExp(x,e) で式 fun x -> e を表す. *)
  | FunExp of id * exp
  (* 関数適用を表す式. AppExp(e1,e2) で式 e1 e2 を表す. *)
  | AppExp of exp * exp
  (* 再帰関数定義を表す式. LetRecExp(f,x,e1,e2) で式 let rec f x = e1 in e2 を表す. *)
  | LetRecExp of id * id * exp * exp

(* 値を表す型 *)
type exval =
    IntV of int (* 整数値 *)
  | BoolV of bool (* 真偽値 *)
  | ProcV of id * exp * dnval Environment.t ref (* 関数閉包 *)
and dnval = exval

exception Error of string
let err s = raise (Error s)

let rec eval_exp env e = match e with
  | ...
  |  Var x ->
      (try Environment.lookup x env with
        Environment.Not_bound -> err ("Variable not bound: " ^ x))
  | IfExp (exp1, exp2, exp3) ->
      let test = eval_exp env exp1 in
        (match test with
            BoolV true -> eval_exp env exp2
          | BoolV false -> eval_exp env exp3
          | _ -> err ("Test expression must be boolean: if"))   (a)
  | FunExp (id, exp) -> ProcV (id, exp, ref env)
  | AppExp (exp1, exp2) ->
      let funval = eval_exp env exp1 in
      let arg = eval_exp env exp2 in
        (match funval with
             ProcV (id, body, env') ->
               let newenv = Environment.extend id arg !env' in
                 eval_exp newenv body
           | _ -> err ("Non-function value is applied"))
  | LetRecExp (id, para, exp1, exp2) ->
      let newenv = Environment.extend id (ProcV (para, exp1, ref env)) env in
            eval_exp newenv exp2
```

図 1　インタプリタの実装の一部.

$$\frac{(\Gamma(x) = \tau)}{\Gamma \vdash x : \tau} \quad \text{(T-Var)}$$

$$\frac{}{\Gamma \vdash n : \mathbf{int}} \quad \text{(T-Int)}$$

$$\frac{(b = \texttt{true} \text{ または } b = \texttt{false})}{\Gamma \vdash b : \mathbf{bool}} \quad \text{(T-Bool)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{int} \qquad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1 + e_2 : \mathbf{int}} \quad \text{(T-Plus)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{int} \qquad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1 * e_2 : \mathbf{int}} \quad \text{(T-Mult)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{int} \qquad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1 < e_2 : \mathbf{bool}} \quad \text{(T-Lt)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{int} \qquad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1 = e_2 : \mathbf{bool}} \quad \text{(T-Eq)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau} \quad \text{(T-If)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : \tau_2} \quad \text{(T-Let)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \texttt{fun } x \to e : \tau_1 \to \tau_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \quad \text{(T-App)}$$

$$\frac{\Gamma, f : \tau_1 \to \tau_2, x : \tau_1 \vdash e_1 : \tau_2 \qquad \Gamma, f : \tau_1 \to \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \texttt{let rec } f = \texttt{fun } x \ \to e_1 \texttt{ in } e_2 : \tau} \quad \text{(T-LetRec)}$$

図 2　$\mathbf{ML}^4$ 言語の型付け規則.