

ocaml 第五期レポート課題

高口 奨一郎
学籍番号：6319045

2020 年 8 月 31 日

1 解いた問題

問題 1,2,3,6,4,5

2 問題 1

SProlog の生成規則と抽象構文木の対応を参考にしながら, SProlog 処理系を完成させなさい

2.1 プログラムソース

```
1 module Lexer = struct
2
3 type token = CID of string | VID of string | NUM of
  string
4           | TO | IS | QUIT | OPEN | EOF | ONE of
  char
5
6 module P = Printf
7
8 exception End_of_system
9
10 let ISTREAM = ref stdin
11
12 let ch = ref []
13
14 let read () = match !ch with [] -> input_char !
  ISTREAM
15                | h::rest -> (ch := rest; h)
16
17 let unread c = ch := c::!ch
18
19 let lookahead () = try let c = read () in unread c; c
  with End_of_file -> '$'
20
21 let rec integer i =
22 (* 文字列として数字を構成 *)
23   let c = lookahead () in
24   if (c >= '0' && c <= '9') then
25     integer (i^(Char.escaped (read ())))
26   else i
27
28 and identifier id =
```

```

29 let c = lookahead () in
30   if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= '
      Z')) ||
31     (c >= '0' && c <= '9') || c == '_' then
32     identifier (id^(Char.escaped (read ())))
33   else id
34
35 and native_token () =
36   let c = lookahead () in
37     if (* CID に対する識別子および予約語 *)
38       (c >= 'a' && c <= 'z') then
39       let id = identifier "" in
40       match id with
41       | "is" -> IS
42       | "quit" -> QUIT
43       | "open" -> OPEN
44       | "eof" -> EOF
45       | _ -> CID (id)
46     else if (* VID に対する識別子 *)
47       (c >= 'A' && c <= 'Z') then VID (identifier "")
48     else if (c >= '0' && c <= '9') then NUM (integer
49       "")
50     else if (* :- を認識して TO を返す *)
51       (c == ':') then (let sub1 = read() in let sub2 =
52       read() in
53       if sub2 == '-' then TO
54       else ONE (sub1))
55     else ONE (read ())
56
57 and gettoken () =
58   try
59     let token = native_token () in
60     match token with
61     | ONE ' ' -> gettoken ()
62     | ONE '\t' -> gettoken ()
63     | ONE '\n' -> gettoken ()
64     | _ -> token
65   with End_of_file -> EOF
66
67 let print_token tk =
68   match tk with
69   | (CID i) -> P.printf "CID(%s)" i
70   | (VID i) -> P.printf "VID(%s)" i

```

```

69 | (NUM i) -> P.printf "NUM(%s)" i
70 | (TO) -> P.printf ":-"
71 | (QUIT) -> P.printf "quit"
72 | (OPEN) -> P.printf "open"
73 | (IS) -> P.printf "is"
74 | (EOF) -> P.printf "eof"
75 | (ONE c) -> P.printf "ONE(%c)" c
76
77 let rec run () =
78     flush stdout;
79     let rlt = gettoken () in
80     match rlt with
81     (ONE '$') -> raise End_of_system
82     | _ -> (print_token rlt; P.printf "\n"; run())
83
84 end
85
86 module Evaluator = struct
87 (* 抽象構文本の型宣言 *)
88 type ast = Atom of string | Var of string | App of
      string * ast list
89
90 (* 抽象構文本の印字関数 *)
91 module P = Printf
92 let rec print_ast ast = match ast with
93 (App(s, hd::tl)) -> (P.printf "App(\"%s\",[" s;
94     print_ast hd; List.iter (fun x -> (print_string
95         ";"; print_ast x)) tl;
96     print_string "]"")
97 | (App(s, [])) -> P.printf "App(\"%s\",[])" s
98 | (Atom s) -> P.printf "Atom \"%s\"" s
99 | (Var s) -> P.printf "Var \"%s\"" s
100 let print_ast_list lst = match lst with
101 (hd::tl) -> (print_string "["; print_ast hd;
102     List.iter (fun x -> (print_string ";";
103         print_ast x)) tl; print_string "]")
104 | [] -> print_string "[]"
105 (* 関数 sub, mgu, succeed, rename solve eval の定義 *)
106 let sub name term =
107     let rec mapVar ast = match ast with
108     (Atom x) -> Atom(x)
109     | (Var n) -> if n=name then term else Var n

```

```

110      | (App(n, terms)) -> App(n, List.map mapVar terms
111    in mapVar
112
113  let mgu (a,b) =
114    let rec ut (one, another, unifier) = match (one,
115      another) with
116      | ([], []) -> (true, unifier)
117      | (term::t1, Var(name)::t2) ->
118        let r = fun x -> sub name term (unifier x) in
119        ut(List.map r t1, List.map r t2, r)
120      | (Var(name)::t1, term::t2) ->
121        let r = fun x -> sub name term (unifier x) in
122        ut(List.map r t1, List.map r t2, r)
123      | (Atom(n)::t1, Atom(m)::t2) ->
124        if n=m then ut(t1,t2,unifier) else (false,
125          unifier)
126      | (App(n1,xt1)::t1, App(n2,xt2)::t2) ->
127        if n1=n2 && List.length xt1 = List.length xt2
128        then
129          ut(xt1@t1, xt2@t2, unifier)
130        else (false, unifier)
131      | (_,_) -> (false, unifier);
132  in ut ([a],[b], (fun x -> x))
133
134  let succeed query = (print_ast query; true)
135
136  let rename ver term =
137    let rec mapVar ast = match ast with
138    | (Atom x) -> Atom(x)
139    | (Var n) -> Var(n^"#"^ver)
140    | (App(n, terms)) -> App(n, List.map mapVar
141      terms)
142  in mapVar term
143
144  exception Compiler_error
145
146  let rec solve (program, question, result, depth) =
147    match question with
148    | [] -> succeed result
149    | goal::goals ->
150      let onestep _ clause =
151        match List.map (rename (string_of_int depth))

```

```

        clause with
147     [] -> raise Compiler_error
148   | head::conds ->
149     let (unifiable, unifier) = mgu(head, goal) in
150       if unifiable then
151         solve (program, List.map unifier (
152           conds@goals),
153               unifier result, depth
154               +1)
155       else true
156   in List.fold_left onestep true program
157
158 let eval (program, question) = solve(program, [
159   question], question, 1)
160
161 end
162
163 module Parser = struct
164   module L = Lexer
165   module E = Evaluator
166   let tok = ref (L.ONE ' ')
167   let getToken () = L.gettoken ()
168   let advance () = (tok := getToken())
169   exception Syntax_error
170   let error () = raise Syntax_error
171   let check t = match !tok with
172     L.CID _ -> if (t = (L.CID "")) then () else error
173     ()
174     | L.VID _ -> if (t = (L.VID "")) then () else
175       error()
176     | L.NUM _ -> if (t = (L.NUM "")) then () else
177       error()
178     | tk -> if (tk=t) then () else error()
179
180 let eat t = (check t; advance())
181
182 let prog = ref [[E.Var ""]]

```

```

185
186 let rec clauses() = match !tok with
187     L.EOF -> []
188     | _ -> let a = clause() in let b = clauses() in a
           :: b
189
190 and clause() = match !tok with
191     L.ONE '(' -> let a = [term()] in eat(L.ONE '.') ;
           a
192     | _ -> let a = predicate() in let b = to_opt() in
           eat(L.ONE '.') ; a :: b
193
194 and to_opt() = match !tok with
195     L.TO -> eat(L.TO) ; let b = terms() in b
196     | _ -> []
197
198 and command() = match !tok with
199     L.QUIT -> exit 0
200     | L.OPEN -> (eat(L.OPEN);
201                  match !tok with
202                      L.CID s -> (eat(L.CID ""); check (L.ONE '.');
203                      L.ISTREAM := open_in (s ^ ".pl"); advance();
204                      prog := clauses(); close_in (!L.ISTREAM))
205     | _ -> error()
206     | _ -> let t = term() in
207             (check(L.ONE '.'); let _ = E.eval(!prog, t)
              in ())
208
209 and term() = match !tok with
210     L.ONE '(' -> (eat(L.ONE '(') ; let b = term() in
211                  eat(L.ONE ')') ; b)
211     | L.VID s -> (eat(L.VID ""); eat(L.IS) ; expr())
212     | _ -> let a = predicate() in a
213
214 and terms() = (let a = [term()] in let c = terms' ()
215                in a @ c)
216
217 and terms' () = match !tok with
218     L.ONE ', ' -> (eat(L.ONE ', ') ; let d = [term()]
219                    in let e = terms'() in d @ e)
220     | _ -> []
221
222 and predicate() = match !tok with

```

```

221      L.CID a -> (eat(L.CID "") ; eat(L.ONE '(') ; let
                b = args() in eat(L.ONE ')') ; E.App (a,b))
222      | _ -> error()
223
224 and args() = let a =[expr()] in let c = args'() in a
                @ c
225
226 and args'() = match !tok with
227      L.ONE ', ' -> (eat(L.ONE ', ') ; let d = [expr()]
                in let e = args'() in d @ e\
228 )
229      | _ -> []
230
231 and expr() = match !tok with
232      L.ONE '(' -> (eat(L.ONE '(') ; let b = expr() in
                eat(L.ONE ')') ; b)
233      | L.ONE '[' -> (eat(L.ONE '[') ; let a = list() in
                eat(L.ONE ']') ; a)
234      | L.CID s -> (eat(L.CID s) ; tail_opt s)
235      | L.VID s -> (eat(L.VID s) ; E.Var s)
236      | L.NUM n -> (eat(L.NUM s) ; E.Atom n)
237      | _ -> error()
238
239 and tail_opt s = match !tok with
240      L.ONE '(' -> (eat(L.ONE '(') ; let a = args() in
                eat(L.ONE ')') ; E.App (s\
241      , a))
242      | _ -> E.Atom s
243
244 and list() = match !tok with
245      L.ONE ']' -> E.Atom "nil"
246      | _ -> E.App ("cons" , [expr() ; list_opt()])
247
248 and list_opt() = match !tok with
249      L.ONE '|' -> (eat(L.ONE '|')) ; id()
250      | L.ONE ', ' -> (eat(L.ONE ', ')) ; list()
251      | _ -> E.Atom "nil"
252
253 and id() = match !tok with
254      L.CID a -> (eat(L.CID s)) ; E.Atom a
255      | L.VID a -> (eat(L.VID s)) ; E.Var a
256      | L.NUM a -> (eat(L.NUM s)) ; E.Atom a
257      | _ -> error()

```



```

258
259 end
260
261 let rec run() =
262     print_string "?- ";
263     while true do
264         flush stdout; Lexer._ISTREAM := stdin;
265         Parser.advance(); Parser.command() ;
266         print_string "\n?- "
267     done
268 let _ = run()

```

2.2 左再帰を除いた文法

新しく `terms'`、`args'` を定義することで左再帰を除き、右再帰に変換することができる。

2.2.1 terms

```

terms → term terms'
terms' → “,” term terms'
terms' →

```

2.2.2 args

```

args → expr args'
args' → “,” expr args'
args' →

```

2.3 プログラムの説明

第四期の課題で作成した Lexer と Parser を用いて、Parser の中身を与えられた文法に従って構成し直した。前回と同様に右再帰の関数は上記のように文法を作り直した。また、Evaluator に関しては授業プリントを参考にしながら作成した。

3 問題 2

isono プログラムを入力し、いくつかの質問について、振舞いを確認しなさい。

3.1 実行結果

以下のように正しく動作していることがわかる。

```
?- open isono.
?- mother(X,Y).
App("mother",[Atom "sazae";Atom "tara"])App("mother",[Atom "fune";Atom "sazae"])
App("mother",[Atom "fune";Atom "katsuo"])App("mother",[Atom "fune";Atom "wakame"
])
?- female(X).
App("female",[Atom "fune"])App("female",[Atom "wakame"])App("female",[Atom "saza
e"])
?-
```

4 問題 3

第 9 章で示した関数 `succeed` では、インスタンス化した質問の構文木を印字するようになっている。これを、SProlog のソース言語の表現で印字するようにしなさい

4.1 プログラムソース

Evaluator 内の抽象構文木について以下のような変更を加えた。

```
1 module P = Printf
2 let rec print_ast ast = match ast with
3   (App(s, hd::tl)) -> (P.printf "%s(" s;
4     print_ast hd; List.iter (fun x -> (print_string ";"; pri
5       print_string ")"))
6   | (App(s, [])) -> P.printf "%s" s
7   | (Atom s) -> P.printf "%s" s
8   | (Var s) -> P.printf "%s" s
9 let print_ast_list lst = match lst with
```

```

10      (hd::tl) -> (print_string "["; print_ast hd;
11          List.iter (fun x -> (print_string ";";
12              print_ast x)) tl; print_string "]")
13      | [] -> print_string "]"

```

4.2 プログラムの説明

抽象構文木の印字関数の部分を Sprolog のソース言語の表現と同じになるように変更を加えた。

4.3 実行結果

以下のように正しく動作していることがわかる。

```

end
val run : unit -> unit = <fun>
?- mother(X,Y).
mother(X;Y)
?-open isono.

?-mother(X,Y).
mother(sazae;tara)mother(fune;sazae)mother(fune;katsuo)mother(fune;wakame)
?-female(X).
female(fune)female(wakame)female(sazae)
?-

```

5 問題 6

[,] で並べた複数質問を受け付けられるように拡張せよ

5.1 プログラムソース

```

1 let rec print a b =
2     match (a, b) with
3     | [], [] -> ()
4     | [], hd :: tl -> ()
5     | hd :: tl, [] -> ()
6     | (hd1 :: tl1, hd2 :: tl2) -> (print_ast hd1 ;
7         print_string "=" ; print_ast hd2; print_str\
8         ing "\n" ; print tl1 tl2)
9 let rec solve (program, question, result, a, b, depth
10     ) =

```

```

10 match question with
11   [] -> succeed result a b
12 | goal :: goals ->
13   let onestep - clause =
14     match List.map (rename (string_of_int depth))
15       clause with
16     | [] -> raise Compiler_error
17     | head :: conds ->
18       let (unifiable, unifier) = mgu(head, goal) in
19         if unifiable then
20           solve (program, List.map unifier (
21             conds@goals),
22               List.map unifier result, a,
23               List.map unifier b, depth +
24                 1)
25         else true
26 in List.fold_left onestep true program
27
28 let rec get x =
29   match x with
30   | [] -> []
31   | hd :: tl -> match hd with
32     | Var t -> Var t :: (get tl)
33     | Atom s -> get tl
34     | App(x,y) -> (get y) @ (get tl)
35
36 let rec find y =
37   let rec sub n lst =
38     match lst with
39     | [] -> []
40     | hd :: tl -> if hd = n then sub n tl
41                     else hd :: sub n tl
42   in match y with
43   | [] -> []
44   | hd :: tl -> hd :: (find (sub hd y))
45
46 let eval (program, question) =
47   let l = find (get question)
48   in solve (program, question, question, l, l, 1)
49
50 and command() = match !tok with
51   | L.QUIT -> exit 0
52   | L.OPEN -> (eat(L.OPEN));
53   match !tok with

```

```

51      L.CID s -> (eat(L.CID "" ); check (L.ONE ' ');
52      L.ISTREAM := open_in (s^".pl"); advance();
53      prog := clauses(); close_in (!L.ISTREAM))
54  | _ -> error() )
55  | _ -> let t = terms() in
56      (check(L.ONE ' '); let _ = E.eval(!prog, t)
        in ())

```

5.2 プログラムの説明

抽象構文木の部分と command 関数に変更を加えることで複数質問に答えられるようにした。

5.3 実行結果

以下のように正しく動作していることがわかる。

```

?-grandfather(X,tara),parents(X,Y).
X=namihei
Y=sazae
X=namihei
Y=katsuo
X=namihei
Y=wakame

```

6 問題 4、問題 5

通常の Prolog は、質問が真であったとき、インスタンス化した質問を印字するのではなく、質問に含まれる変数ごとに、対応する項を印字する。SProlog の処理系もそのように拡張せよ

通常の Prolog は、質問が真である単一化代入が得られるたびに、推論を継続するか終了するかを指示することができる。SProlog の処理系も、succeed が呼ばれるたびに、実行を中断し、[;] を打ち込むと継続、[.] を打ち込むと、それ以降の処理を回避するように拡張しなさい

6.1 プログラムソース

```

1 let succeed query a b =
2 (print a b ; print_string "Yes"; flush stdout ;

```

```

3 let y = ref ' ' in while (!y != ';' && !y != '.') do
4 y := input_char stdin done; flush stdout;
5 if !y = ';' then true else raise Error)
6 let rec run() =
7   print_string "?- ";
8   while true do
9     flush stdout; Lexer.ISTREAM := stdin;
10    Parser.advance(); (try Parser.command() with
11      Evaluator.Error ->
12    print_string "No"); print_string "\n?- "

```

6.2 プログラムの説明

問題 6 から印字関数に手を加えることで表記を正しいものに変更した。

6.3 実行結果

以下のように正しく動作していることがわかる。



```

?- parents(X,tara).
X=masuo
Yes;
X=sazae
Yes.
No
?-

```