情報科学実験 sprolog 処理系の完成

６３２１１２０
横溝尚也

提出日：８月９日（火）

回答した問題：課題５問題１，２，３，４
ソースコード名
問題１：spl_syorikei_1.ml
問題３：spl_syorieki_3.ml
問題４：spl_syorikei_4.ml

1　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.
自分は課題４（prolog の字句解析と構文解析）で文法に算術式を追加するために構文解析器を arithexp に拡張してあるが、一旦 expr に直し、四則演算に対応していない構文解析器で問題に取り組んだ。

1.1　プログラムの主な変更点

変更点は構文解析器である。
・126 行目（以下のプログラムに行数はふってある）に資料に関数 prog を追加
・129 行目の L.print_token の部分をコメントアウトした。
・138〜203 行目の基本文法の処理をすべて変更
・215 行目に関数を追加
問題１のプログラムは以下である。

```
1module Evaluator =
2struct
3  type ast = |Atom of string
4             |Var of string
5             |App of string * ast list

6  module P = Printf

7  exception Compiler_error

8  let rec print_ast ast =
9    match ast with
10    |(App(s, hd::tl)) -> (P.printf "App(\"%s\",[" s ; print_ast hd;
List.iter (fun x -> (print_string ";"; print_ast x)) tl; print_string "])")
11    |(App(s, [])) -> P.printf "App(\"%s\",[])" s
12    |(Atom s) -> P.printf "Atom \"%s\"" s
13    |(Var s) -> P.printf "Var \"%s\"" s

14  let print_ast_list lst =
15    match lst with
16    |(hd::tl) -> (print_string "["; print_ast hd; List.iter (fun x ->
(print_string ";"; print_ast x))tl; print_string "]")
17    |[] -> print_string "[]"

18  let sub name term =
```

```
19    let rec mapVar ast = match ast with
20      |(Atom x) -> Atom(x)
21      |(Var n) -> if n=name then term else Var n
22      |(App(n, terms)) -> App(n, List.map mapVar terms)
23    in mapVar

24  let mgu (a,b) =
25    let rec ut (one, another, unifier) =
26      match (one, another) with
27      |([], []) -> (true, unifier)
28      |(term::t1, Var(name)::t2) ->
29          let r = fun x -> sub name term (unifier x) in
31          ut(List.map r t1, List.map r t2, r)
32      |(Var(name)::t1, term::t2) ->
33          let r = fun x -> sub name term (unifier x) in
34          ut(List.map r t1, List.map r t2, r)
35      |(Atom(n)::t1, Atom(m)::t2) ->
36          if n=m then ut(t1,t2,unifier) else (false, unifier)
37      |(App(n1,xt1)::t1, App(n2,xt2)::t2) ->
38          if n1=n2 && List.length xt1 = List.length xt2 then
39            ut(xt1@t1, xt2@t2, unifier)
40          else (false, unifier)
41      |(_,_) -> (false, unifier);
42    in ut ([a],[b], (fun x -> x))

43  let succeed query = (print_ast query; true)

44  let rename ver term =
45    let rec mapVar ast =
46      match ast with
47      |(Atom x) -> Atom(x)
48      |(Var n) -> Var(n^"#"^ver)
49      |(App(n, terms)) -> App(n, List.map mapVar terms)
50    in mapVar term

51  let rec solve (program, question, result, depth) =
52    match question with
53    |[] -> succeed result
54    |goal::goals ->
55        let onestep _ clause =
```

```
56          match List.map (rename (string_of_int depth)) clause with
57          |[] -> raise Compiler_error
58          |head::conds ->
59              let (unifiable, unifier) = mgu(head,goal) in
60              if unifiable then
61                solve (program, List.map unifier (conds@goals), unifier result, depth+1)
62              else true
63        in List.fold_left onestep true program

64  let eval (program, question) = solve(program, [question], question, 1)
65end ;;

(*字句解析*)
66module Lexer = struct
67  type token = CID of string | VID of string | NUM of string
68             | TO | IS | QUIT | OPEN | EOF | ONE of char
69  module P = Printf
70  exception End_of_system
71  let count  = ref 1
72  let _ISTREAM = ref stdin
73  let ch = ref []
74  let read () = match !ch with [] -> input_char !_ISTREAM
75                               | h::rest -> (ch := rest; h)
76  let unread c = ch := c::!ch
77  let lookahead () = try let c = read () in unread c; c with End_of_file -> '$'
78  let rec integer i =
79    let c = lookahead () in
80    if (c >= '0' && c <= '9') then
81      integer (i^(Char.escaped (read ())))
82    else i
83  and identifier id =
84    let c = lookahead () in
85    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||
86        (c >= '0' && c <= '9') || c == '_') then
87      identifier (id^(Char.escaped (read ())))
88    else id
89  and native_token () =
90    let c = lookahead () in
91    if (c >= 'a' && c <= 'z') then
92      let id = identifier "" in
```

```
93      match id with
94        "is" -> IS
95      | "quit" -> QUIT
96      | "open" -> OPEN
97      | _ -> CID (id)
98    else if (c >= 'A' && c <= 'Z') then VID (identifier "")
99    else if (c >= '0' && c <= '9') then NUM (integer "")
100    else if (c = ':') then (read() ; if (lookahead()=='-') then (read(); TO) else ONE (c))
101    else ONE (read ())
102  and gettoken () =
103    try
104      let token = native_token () in
105      match token with
106        ONE ' ' ->  gettoken ()
107      | ONE '\t' -> gettoken ()
108      | ONE '\n' -> count := !count + 1 ; gettoken ()
109      |  _ ->  token
110    with End_of_file -> EOF

111  let print_token tk =
112    match tk with
113      (CID i) -> P.printf "CID(%s)" i
114    | (VID i) -> P.printf "VID(%s)" i
115    | (NUM i) -> P.printf "NUM(%s)" i
116    | (TO) -> P.printf ":-"
117    | (QUIT) -> P.printf "quit"
118    | (OPEN) -> P.printf "open"
119    | (IS) -> P.printf "is"
120    | (EOF) -> P.printf "eof"
121    | (ONE c) -> P.printf "ONE(%c)" c
122end
```

(*構文解析*)

```
123module Parser = struct
124  module L = Lexer
125  module E = Evaluator
126  let prog = ref [[E.Var ""]]   (*追加*)
127  let tok = ref (L.ONE ' ')
128  let getToken () = L.gettoken ()
129  let advance () = (tok := getToken();(* L.print_token (!tok)*))
```

```
(*L.print_token (!tok) をコメントアウトした*)
130  exception Syntax_error
131  let error () = raise Syntax_error
132  let check t = match !tok with
133      L.CID _-> if (t = (L.CID "")) then () else error()
134    | L.VID _-> if (t = (L.VID "")) then () else error()
135    | L.NUM _-> if (t = (L.NUM "")) then () else error()
136    | tk -> if (tk=t) then () else error()
137  let eat t = (check t; advance())
138  let rec clauses() = match !tok with
139      L.EOF -> () ; []
140    | _  -> (let c1 = clause() in
141             let c2 = clauses() in
142             (c1 :: c2))
143  and clause() = match !tok with
144      L.ONE '(' -> let t1 = term() in eat(L.ONE '.') ;[t1]
145    | _ ->let t1 = predicate() in
146        let t2 = to_opt()
147        in eat(L.ONE '.') ; (t1::t2)
148  and to_opt() = match !tok with
149      L.TO -> (eat(L.TO);let t1 = terms() in (t1))
150    | _ -> []
151  and command() = match !tok with
152      L.QUIT -> exit 0
153    | L.OPEN -> (eat(L.OPEN);
154                 match !tok with
155                   L.CID s -> (eat(L.CID ""); check (L.ONE '.');
156                               L._ISTREAM := open_in (s^".pl"); advance();
157                               prog := clauses(); close_in (!L._ISTREAM))
158                 | _ -> error())
159    | _ -> let t = term() in (check(L.ONE '.'); let _ = E.eval(!prog, t) in ())
160  and term() = match !tok with
161      L.ONE '(' -> eat(L.ONE '(');
162        let u1 = term() in eat(L.ONE '('); (u1)
163    | _ -> predicate()
164  and terms() = let u1 = term() in let u2 = terms'() in [u1]@u2
165  and terms'() = match !tok with
166      L.ONE ',' ->  eat(L.ONE ',');
167        let u1 = term() in let u2 = terms'() in
168        [u1]@u2
```

```
169    | _ -> []
170  and predicate() = match !tok with
171    |L.CID d -> (eat(L.CID "") ; eat(L.ONE '(');
172                 let s1 = args() in eat(L.ONE ')');
173                 E.App(d,s1))
174    |_ -> error ()
175  and args() = let ext1 = expr() in let s1 = args'() in [ext1]@s1
176  and args'() = match !tok with
177      L.ONE ','->  eat(L.ONE ',');
178        let ext1 = expr() in let s1 = args'() in
179        [ext1]@s1
180    | _ -> []
181  and expr() = match !tok with
182      L.ONE '(' -> eat(L.ONE '(');let ext1 = expr() in eat(L.ONE ')');ext1
183    |L.ONE '[' -> eat(L.ONE '[');let list1 = list() in eat(L.ONE ']');list1
184    |L.CID s -> eat(L.CID "");let tail1 = tail_opt(s) in tail1
185    |L.VID s -> eat(L.VID "");E.Var s
186    |L.NUM n -> eat(L.NUM "");E.Atom n
187    | _ -> error()
188  and tail_opt s =  match !tok with
189      L.ONE '(' -> eat(L.ONE '(') ; let arg1 = args() in eat(L.ONE ')') ; E.App (s,arg1)
190    |_ -> E.Atom s
191  and list() = match !tok with
192      L.ONE ']'-> E.Atom "nil"
193    | _ -> let ext1 = expr() in
194        let l1 = list_opt() in E.App("cons",[ext1;l1])
195  and list_opt() = match !tok with
196      L.ONE '|' -> eat(L.ONE '|');let i1 = id() in i1
197    | L.ONE ',' -> eat(L.ONE ','); let p1 = list() in p1
198    | _ -> E.Atom "nil"
199  and id() = match !tok with
200      L.CID s -> eat(L.CID ""); E.Atom s
201    | L.VID s -> eat(L.VID ""); E.Var s
202    | L.NUM n -> eat(L.NUM ""); E.Atom n
203    | _ -> error ()
204end

205let rec run() =
206  print_string "?- ";
207  while true do
```

```
208    flush stdout; Lexer._ISTREAM := stdin;
209    Parser.advance(); Parser.command(); print_string "\n?- "
210  done

211let run' () =
212  try let c = run () in c
213  with  Parser.Syntax_error -> print_string "\n"; print_string "文法エラー:" ;
Printf.printf ("%d") !Lexer.count; print_string "行目"; 214print_string "\n?-"

215let _ = run'()
```

1.2 変更点について

関数 prog, や 215 行目の関数の追加、comannd 関数の拡張については授業資料に解説が載っているため省略する。129 行目の L.print_token をコメントアウトした理由は、今までのプログラムでは CID(father) のように構文解析した後の type を一つ一つ出力するように字句解析の print_token 関数で定めている。しかし、実際の spl ではそのような出力はない。何か入力したら、それを事実として認識するだけで何も出力されないはずである。そのため、今まで出力していた print_token 関数を使用している 129 行目を無効化（コメントアウト）いた。

SProlog の各生成規則に対応した抽象構文木を生成するために既存の文法処理を変更していった。（これを一つ一つ解説するのは長すぎると判断しました。）

実行結果については問題 2 でふるまいを確認するため省略する。

2　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.
問題１で作成したプログラムを用いて isono.pl をファイル入力したときの実行結果は以下である。

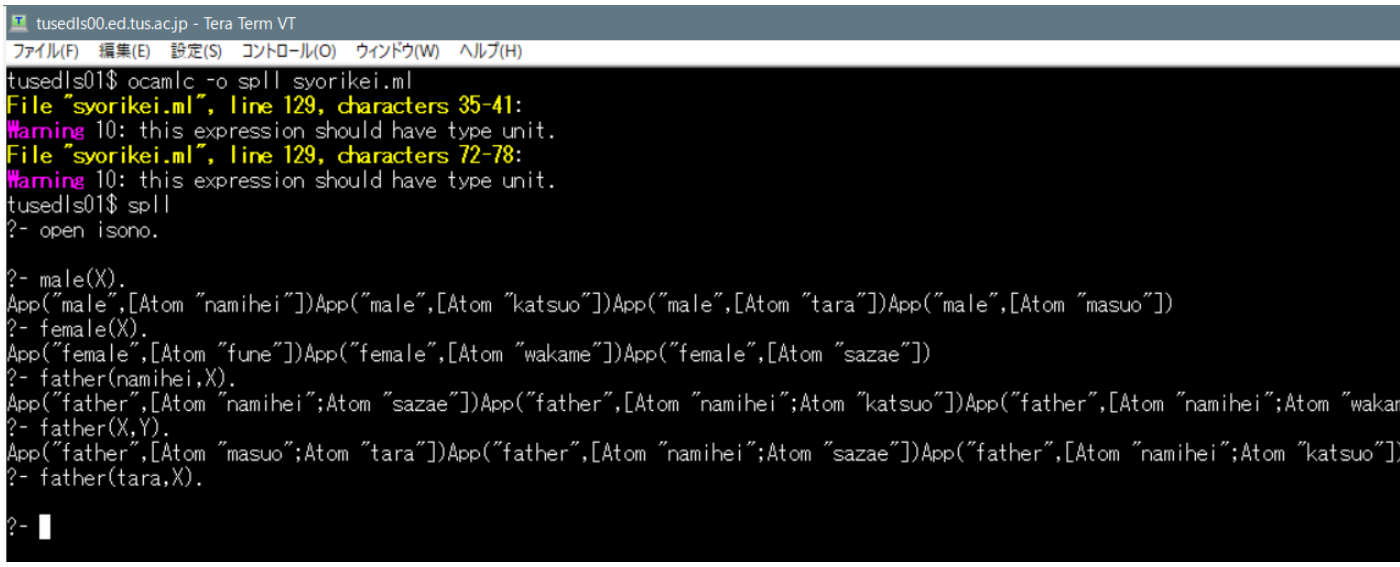

図 1　isno.pl のふるまい確認

3 問題３

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.

3.1 主な変更点

・推論エンジン 50,51 行目の間に独自に作成した print_succeed 関数と、既存の succeed 関数の拡張を行った。これを a1～a11 行目とした。
以下が問題２のプログラムである。

```
1    module Evaluator =
2struct
3  type ast = |Atom of string
4             |Var of string
5             |App of string * ast list
6  module P = Printf
7  exception Compiler_error
8  let rec print_ast ast =
9    match ast with
10    |(App(s, hd::tl)) -> (P.printf "App(\"%s\",[" s ; print_ast hd; List.iter (fun x -> (print_str
11    |(App(s, [])) -> P.printf "App(\"%s\",[])" s
12    |(Atom s) -> P.printf "Atom \"%s\"" s
13    |(Var s) -> P.printf "Var \"%s\"" s

14  let print_ast_list lst =
15    match lst with
16    |(hd::tl) -> (print_string "["; print_ast hd; List.iter (fun x ->
(print_string ";"; print_ast x))tl; print_string "]")
17    |[] -> print_string "[]"

18  let sub name term =
19    let rec mapVar ast = match ast with
20      |(Atom x) -> Atom(x)
21      |(Var n) -> if n=name then term else Var n
22      |(App(n, terms)) -> App(n, List.map mapVar terms)
23    in mapVar

24  let mgu (a,b) =
25    let rec ut (one, another, unifier) =
```

```
26      match (one, another) with
27      |([], []) -> (true, unifier)
28      |(term::t1, Var(name)::t2) ->
29          let r = fun x -> sub name term (unifier x) in
31          ut(List.map r t1, List.map r t2, r)
32      |(Var(name)::t1, term::t2) ->
33          let r = fun x -> sub name term (unifier x) in
34          ut(List.map r t1, List.map r t2, r)
35      |(Atom(n)::t1, Atom(m)::t2) ->
36          if n=m then ut(t1,t2,unifier) else (false, unifier)
37      |(App(n1,xt1)::t1, App(n2,xt2)::t2) ->
38          if n1=n2 && List.length xt1 = List.length xt2 then
39            ut(xt1@t1, xt2@t2, unifier)
40          else (false, unifier)
41      |(_,_) -> (false, unifier);
42    in ut ([a],[b], (fun x -> x))

44  let rename ver term =
45    let rec mapVar ast =
46      match ast with
47      |(Atom x) -> Atom(x)
48      |(Var n) -> Var(n^"#"^ver)
49      |(App(n, terms)) -> App(n, List.map mapVar terms)
50    in mapVar term

a1    let rec print_succeed lst= (*3*)
a2    match lst with
a3    |[] ->P.printf ")"
a4    |(Atom s)::rest -> P.printf (",%s") s; print_succeed rest
a5    | _ -> ()

a6    let succeed query =  (*3*)
a7     let rec succeed2 query = match query with
a8       |App (_ , []) ->  () ;
a9       |App (predicate , (Atom s)::rest ) -> P.printf ("%s") predicate;
P.printf "(" ; P.printf ("%s") s ; print_succeed rest
a10      | _ -> ()
a11    in succeed2 query ; true

51  let rec solve (program, question, result, depth) =
```

```
52    match question with
53    |[] -> succeed result
54    |goal::goals ->
55        let onestep _ clause =
56          match List.map (rename (string_of_int depth)) clause with
57          |[] -> raise Compiler_error
58          |head::conds ->
59              let (unifiable, unifier) = mgu(head,goal) in
60              if unifiable then
61                solve (program, List.map unifier (conds@goals), unifier result, depth+1)
62              else true
63        in List.fold_left onestep true program

64  let eval (program, question) = solve(program, [question], question, 1)
65end ;;



(*字句解析*)
66module Lexer = struct
67  type token = CID of string | VID of string | NUM of string
68             | TO | IS | QUIT | OPEN | EOF | ONE of char
69  module P = Printf
70  exception End_of_system
71  let count  = ref 1
72  let _ISTREAM = ref stdin
73  let ch = ref []
74  let read () = match !ch with [] -> input_char !_ISTREAM
75                               | h::rest -> (ch := rest; h)
76  let unread c = ch := c::!ch
77  let lookahead () = try let c = read () in unread c; c with End_of_file -> '$'
78  let rec integer i =
79    let c = lookahead () in
80    if (c >= '0' && c <= '9') then
81      integer (i^(Char.escaped (read ())))
82    else i
83  and identifier id =
84    let c = lookahead () in
85    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||
86        (c >= '0' && c <= '9') || c == '_') then
```

```
87      identifier (id^(Char.escaped (read ())))
88    else id
89  and native_token () =
90    let c = lookahead () in
91    if (c >= 'a' && c <= 'z') then
92      let id = identifier "" in
93      match id with
94        "is" -> IS
95      | "quit" -> QUIT
96      | "open" -> OPEN
97      | _ -> CID (id)
98    else if (c >= 'A' && c <= 'Z') then VID (identifier "")
99    else if (c >= '0' && c <= '9') then NUM (integer "")
100    else if (c = ':') then (read() ; if (lookahead()=='-') then (read(); TO) else ONE (c))
101    else ONE (read ())
102  and gettoken () =
103    try
104      let token = native_token () in
105      match token with
106        ONE ' ' ->  gettoken ()
107      | ONE '\t' -> gettoken ()
108      | ONE '\n' -> count := !count + 1 ; gettoken ()
109      |  _ ->  token
110    with End_of_file -> EOF

111  let print_token tk =
112    match tk with
113      (CID i) -> P.printf "CID(%s)" i
114    | (VID i) -> P.printf "VID(%s)" i
115    | (NUM i) -> P.printf "NUM(%s)" i
116    | (TO) -> P.printf ":-"
117    | (QUIT) -> P.printf "quit"
118    | (OPEN) -> P.printf "open"
119    | (IS) -> P.printf "is"
120    | (EOF) -> P.printf "eof"
121    | (ONE c) -> P.printf "ONE(%c)" c
122end
```

(*構文解析*)

```
123module Parser = struct
```

```
124 module L = Lexer
125 module E = Evaluator
126 let prog = ref [[E.Var ""]]
127 let tok = ref (L.ONE ' ')
128 let getToken () = L.gettoken ()
129 let advance () = (tok := getToken();(* L.print_token (!tok)*))
130 exception Syntax_error
131 let error () = raise Syntax_error
132 let check t = match !tok with
133     L.CID _-> if (t = (L.CID "")) then () else error()
134   | L.VID _-> if (t = (L.VID "")) then () else error()
135   | L.NUM _-> if (t = (L.NUM "")) then () else error()
136   | tk -> if (tk=t) then () else error()
137 let eat t = (check t; advance())
138 let rec clauses() = match !tok with
139     L.EOF -> () ; []
140   | _  -> (let c1 = clause() in
141            let c2 = clauses() in
142            (c1 :: c2))
143 and clause() = match !tok with
144     L.ONE '(' -> let t1 = term() in eat(L.ONE '.') ;[t1]
145   | _ ->let t1 = predicate() in
146       let t2 = to_opt()
147       in eat(L.ONE '.') ; (t1::t2)
148 and to_opt() = match !tok with
149     L.TO -> (eat(L.TO);let t1 = terms() in (t1))
150   | _ -> []
151 and command() = match !tok with
152     L.QUIT -> exit 0
153   | L.OPEN -> (eat(L.OPEN);
154                match !tok with
155                  L.CID s -> (eat(L.CID ""); check (L.ONE '.');
156                              L._ISTREAM := open_in (s^".pl"); advance();
157                               prog := clauses(); close_in (!L._ISTREAM))
158                | _ -> error())
159   | _ -> let t = term() in (check(L.ONE '.'); let _ = E.eval(!prog, t) in ())
160 and term() = match !tok with
161     L.ONE '(' -> eat(L.ONE '(');
162       let u1 = term() in eat(L.ONE ')'); (u1)
163   | _ -> predicate()
```

```
164 and terms() = let u1 = term() in let u2 = terms'() in [u1]@u2
165 and terms'() = match !tok with
166     L.ONE ',' ->  eat(L.ONE ',');
167       let u1 = term() in let u2 = terms'() in
168       [u1]@u2
169   | _ -> []
170 and predicate() = match !tok with
171   |L.CID d -> (eat(L.CID "") ; eat(L.ONE '(');
172                let s1 = args() in eat(L.ONE ')');
173                E.App(d,s1))
174   |_ -> error ()
175 and args() = let ext1 = expr() in let s1 = args'() in [ext1]@s1
176 and args'() = match !tok with
177     L.ONE ','->  eat(L.ONE ',');
178       let ext1 = expr() in let s1 = args'() in
179       [ext1]@s1
180   | _ -> []
181 and expr() = match !tok with
182     L.ONE '(' -> eat(L.ONE '(');let ext1 = expr() in eat(L.ONE ')');ext1
183   |L.ONE '[' -> eat(L.ONE '[');let list1 = list() in eat(L.ONE ']');list1
184   |L.CID s -> eat(L.CID "");let tail1 = tail_opt(s) in tail1
185   |L.VID s -> eat(L.VID "");E.Var s
186   |L.NUM n -> eat(L.NUM "");E.Atom n
187   | _ -> error()
188 and tail_opt s =  match !tok with
189     L.ONE '(' -> eat(L.ONE '(') ; let arg1 = args() in eat(L.ONE ')') ; E.App (s,arg1)
190   |_ -> E.Atom s
191 and list() = match !tok with
192     L.ONE ']'-> E.Atom "nil"
193   | _ -> let ext1 = expr() in
194       let l1 = list_opt() in E.App("cons",[ext1;l1])
195 and list_opt() = match !tok with
196     L.ONE '|' -> eat(L.ONE '|');let i1 = id() in i1
197   | L.ONE ',' -> eat(L.ONE ','); let p1 = list() in p1
198   | _ -> E.Atom "nil"
199 and id() = match !tok with
200     L.CID s -> eat(L.CID ""); E.Atom s
201   | L.VID s -> eat(L.VID ""); E.Var s
202   | L.NUM n -> eat(L.NUM ""); E.Atom n
203   | _ -> error ()
```

```
204end

205let rec run() =
206  print_string "?- ";
207  while true do
208    flush stdout; Lexer._ISTREAM := stdin;
209    Parser.advance(); Parser.command(); print_string "\n?- "
210  done

211let run' () =
212  try let c = run () in c
213  with  Parser.Syntax_error -> print_string "\n"; print_string "文法エラー:" ;
Printf.printf ("%d") !Lexer.count; print_string "行目"; 214print_string "\n?-"

215let _ = run'()
```

3.2 変更点について

a1〜a5 行目の補助関数 print_succeed 関数の作成と、a6〜a11 行目の succeed 関数の拡張について説明する。まずこの問題では問題 2 の時点で推論された結果を抽象構文木で出力されていた。それを必要な箇所だけ、出力するようにプログラムをした。つまり、出力する際に引数として受け取る query をそのまま出力するのではなく、match 文によって抽象構文木を分解して father などを表す predicate、sazae などを表す Atom s の s の部分のみを出力するように succeed 関数を拡張した。また、father(masuo,X) などの複数の項からなる質問に対しても、同様の処理を行いたいため、補助関数 print_succeed を宣言し 2 番目の項も出力できるようにした。

実行結果は以下のようになる。

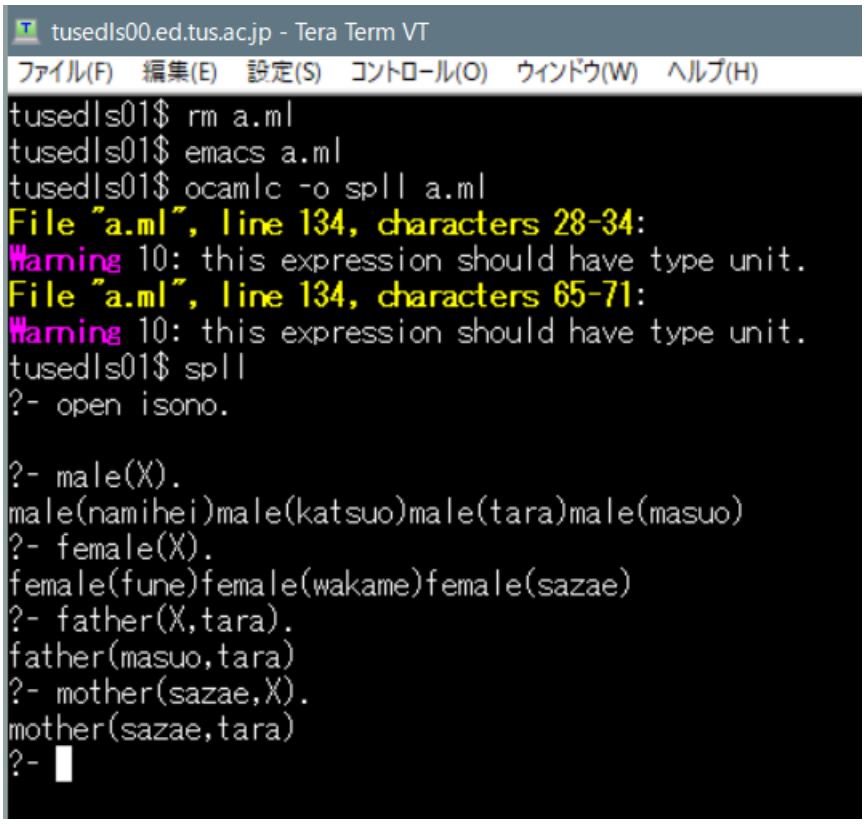
tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls01$ rm a.ml
tusedls01$ emacs a.ml
tusedls01$ ocamlc -o spll a.ml
File "a.ml", line 134, characters 28-34:
Warning 10: this expression should have type unit.
File "a.ml", line 134, characters 65-71:
Warning 10: this expression should have type unit.
tusedls01$ spll
?- open isono.
?- male(X).
male(namihei)male(katsuo)male(tara)male(masuo)
?- female(X).
female(fune)female(wakame)female(sazae)
?- father(X,tara).
father(masuo,tara)
?- mother(sazae,X).
mother(sazae,tara)
?-

図 2　問題 4 実行結果

4 問題４

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．

最後に問題４までのプログラムを添付する。ここでは主に推論エンジンの変更場所や、追加場所と、構文解析の command 関数を変更したのでそこだけ記す。

```
module Evaluator =
struct
  type ast = |Atom of string
             |Var of string
             |App of string * ast list

  module P = Printf

  exception Compiler_error

  let rec print_ast ast =
    match ast with
    |(App(s, hd::tl)) -> (P.printf "App(\"%s\",[" s ; print_ast hd;
    List.iter (fun x -> (print_string ";"; print_ast x)) tl; print_string "])")
    |(App(s, [])) -> P.printf "App(\"%s\",[])" s
    |(Atom s) -> P.printf "Atom \"%s\"" s
    |(Var s) -> P.printf "Var \"%s\"" s

  let print_ast_list lst =
    match lst with
    |(hd::tl) -> (print_string "["; print_ast hd; List.iter
    (fun x -> (print_string ";"; print_ast x)) tl; print_string "]")
    |[] -> print_string "[]"

  let save_hensuu = ref (Var "") and save_hensuu' = ref "" and save_atom = ref ""
(*追加*)
  let sub name term = (*変更*)
    let rec mapVar ast = match ast with
      |(Atom x) -> save_atom := x ; Atom(x)
      |(Var n) -> (save_hensuu := (Var n); if n=name then term else Var n)
```

```
    |(App(n, terms)) -> App(n, List.map mapVar terms)
  in mapVar

let mgu (a,b) =

  let rec ut (one, another, unifier) =
    match (one, another) with

    |([], []) -> (true, unifier)

    |(term::t1, Var(name)::t2) ->
        let r = fun x -> sub name term (unifier x) in
        ut(List.map r t1, List.map r t2, r)

    |(Var(name)::t1, term::t2) ->
        let r = fun x -> sub name term (unifier x) in
        ut(List.map r t1, List.map r t2, r)

    |(Atom(n)::t1, Atom(m)::t2) ->
        if n=m then ut(t1,t2,unifier) else (false, unifier)

    |(App(n1,xt1)::t1, App(n2,xt2)::t2) ->
        if n1=n2 && List.length xt1 = List.length xt2 then
          ut(xt1@t1, xt2@t2, unifier)
        else (false, unifier)

    |(_,_) -> (false, unifier);
  in ut ([a],[b], (fun x -> x))

let rename ver term =
  let rec mapVar ast =
    match ast with
    |(Atom x) -> Atom(x)
    |(Var n) -> Var(n^"#"^ver)
    |(App(n, terms)) -> App(n, List.map mapVar terms)
  in mapVar term


let count_of_var = ref 0 (*追加*)
```

```
let rec  print_succeed lst=  (*変更*)
  match lst with
  |[] ->let ex1 () = P.printf ")" and ex2 () = P.printf " " in
      if !count_of_var = 0 then ex1 ()  else ex2 ()
  |(Atom s)::rest -> let ex1 () = P.printf (",%s") s; print_succeed rest
  and ex2 () = P.printf ("%s ; ")s; print_succeed rest in
      if !count_of_var = 0 then ex1 () else ex2 ()
  | _ -> ()

let test_variable x = (*追加*)
  if x == !save_hensuu' then true
  else false

let rec succeed query = match !save_hensuu with (*変更*)
  |(Var "") ->
      let rec succeed' query = match query with
        |App (_ , []) ->  () ;
        |App (predicate , [Atom s]) -> let ex1 () = P.printf ("%s") predicate; P.printf "(" ;
        P.printf ("%s)") s
            and ex2 () = P.printf ("%s ; ") s in
            if !count_of_var = 0 then ex1 ()
            else ex2 ()
        |App (predicate , (Atom s)::rest ) -> let ex0 () = P.printf ("%s ; ") s ;
        print_succeed rest and ex0' () = P.printf ""; print_succeed rest in
            let ex1 () = P.printf "(" ; P.printf ("%s") s ; print_succeed rest
            and ex2 () = if !save_atom == s then ex0' () else ex0  ()  in
            if !count_of_var = 0 then ex1 ()
            else ex2 ()
        | _ -> ()
      in succeed' query ; true
  |(Var  n) -> (match test_variable n with
      |true  -> save_hensuu := (Var ""); succeed query
      |false -> P.printf ("%s = ") n;
      save_hensuu' := n; save_hensuu := (Var ""); count_of_var := 1;  succeed query    )
  | _ -> false

let rec solve (program, question, result, depth) =
  match question with
```

```
    |[] -> succeed result
    |goal::goals ->
        let onestep _ clause =
          match List.map (rename (string_of_int depth)) clause with
          |[] -> raise Compiler_error
          |head::conds ->
              let (unifiable, unifier) = mgu(head,goal) in
              if unifiable then
                solve (program, List.map unifier (conds@goals), unifier result, depth+1)
              else true
        in List.fold_left onestep true program



  let eval (program, question) = solve(program, [question], question, 1)
end ;;
```

```
  and command() = L.count := 1 ; E.count_of_var := 0; (*変更*)
    match !tok with
      L.QUIT -> exit 0
    | L.OPEN -> (eat(L.OPEN);
                 match !tok with
                   L.CID s -> (eat(L.CID ""); check (L.ONE '.');
                               L._ISTREAM := open_in (s^".pl"); advance();
                               prog := clauses(); close_in (!L._ISTREAM))
                 | _ -> error())
    | _ -> let t = term() in (check(L.ONE '.'); let _ = E.eval(!prog, t) in ())
```

4.0.1 変更点について

主に推論エンジンを変更した。初めに save_hensuu 関数を作成した。これは表示したい var,atom を保管しておく関数として機能している。次に count_of_var 関数を作成した。これは spl に入力した際に変数が含まれているのかどうかを判別して、0 または 1 としてほかの関数に使用していくためのものである。

最後に test_variable 関数は、出力する際、変数 X を満たすものが複数個存在したときに、X =〜となるように X が毎回出力されないようにするための関数である。

以上の 3 つの関数を主に追加し、これらの関数を推論エンジンで使用していくために sub,print_succeed,succeed, command(構文解析の中) 4 つの関数の中身を変更した。

すべての関数の説明をするととても長くなるためこのレポートでは結果を示す。以下が実行結果である。

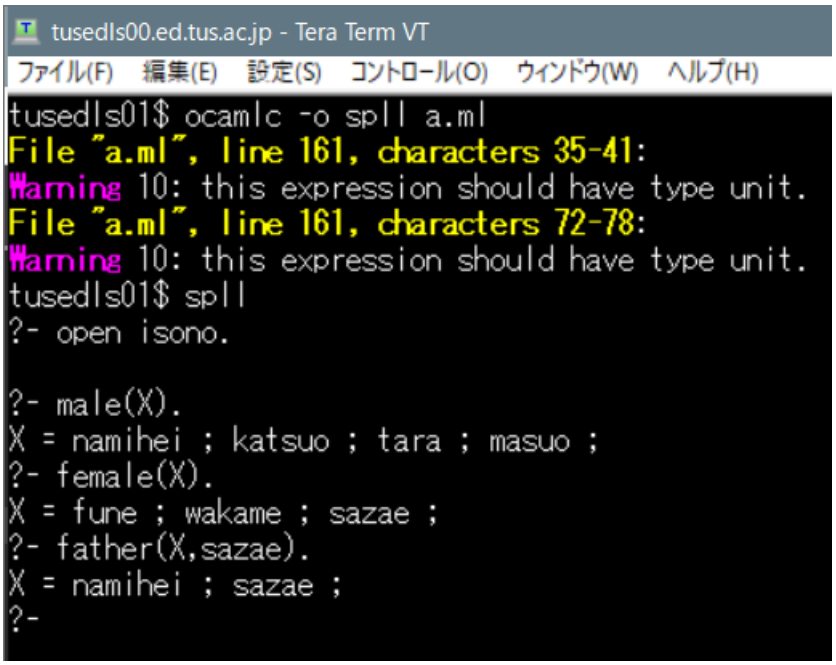
tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls01$ ocamlc -o spll a.ml
File "a.ml", line 161, characters 35-41:
Warning 10: this expression should have type unit.
File "a.ml", line 161, characters 72-78:
Warning 10: this expression should have type unit.
tusedls01$ spll
?- open isono.
?- male(X).
X = namihei ; katsuo ; tara ; masuo ;
?- female(X).
X = fune ; wakame ; sazae ;
?- father(X,sazae).
X = namihei ; sazae ;
?-

図 3　問題 5 実行結果