TP1 - Analyse lexicale et crible

Hassan EL OMARI ALAOUI - Julien MARCHAIS

September 17, 2014

1 Questions

1.1 Question 1.1

Séparer le crible dans l'analyseur lexical permet d'avoir un analyseur lexical plus générique. Toutes les unités lexicales sont réunies au sein de la même fonction. La maintenance de cet analyseur lexical en est donc facilitée.

1.2 Question 1.2

Il est possible de reconnaître les mots-clés "et" et "ou" par le même lexème que ident. Cependant, cela aura pour impact de modifier l'interpretation du crible. En effet, nous n'aurons plus UL_ET et UL_OU mais UL_IDENT "et" et UL_IDENT "ou". De plus, cela diminuera le nombre d'états de l'AFD en compactant ces trois états en un seul.

1.3 Question 1.3

Les types énumérés en Ocaml permettent de réaliser un filtrage sur plusieurs constructeurs. Cela permet d'être plus explicite dans les valeurs retournées et d'éviter les erreurs de typage. Le code est plus optimisé, plus performant.

1.4 Question 1.4

L'intérêt de faire un scanner ne rendant qu'un lexème à la fois permet de simplifier l'implémentation de l'analyseur lexical. En effet, il est plus simple de rendre un seul lexème que de parcourir toute une liste.

1.5 Question 1.5

ident est un terminal car il est utilisé comme une unité lexicale.

1.6 Question 1.6

Il faut ajouter des règles pour les token "si", "sinon", "fsi" et les deux opérateurs de comparaison (fichier lexer.mll). De plus, il faut ajouter dans ulex.ml les token correspondants ainsi que leur message de sortie.

2 Code source

Listing 1: main.ml

```
open Ulex

(** scanner : lexbuf -> token list
    scanner lexbuf constructs the list of token read from the argument
       lexbuf
    (ends when reading UL_EOF )
*)
let scanner lexbuf =

  let rec scanner_rec n l =
    try
      match Lexer.token lexbuf with
      | UL_EOF as tk ->  (tk::l)
      |   tk    -> scanner_rec (n+1) (tk :: l)
  with x ->
    begin
    Printf.printf "Warning : exception %s was raised after reading %i
       tokens\n"
      (Printexc.to_string x) n;
      l
    end
  in
    List.rev (scanner_rec 0 []);;



let _ =
  (** 1. Construct a lex buffer from the standard input channel *)
  let lexbuf = Lexing.from_channel stdin in

  (** 2. Construct the list of tokens *)
  let tokens = scanner lexbuf in

  (** 3. Print the tokens *)
  List.iter (fun tk -> Printf.fprintf stdout "Token: %a\n"
     Ulex.print_token tk) tokens ;
  Printf.printf "DONE\n"
```

Listing 2: ulex.ml

```
(** [token] is the type of the different lexical units. *)
type token =
              | UL_EOF
       | UL_PARFERM
       | UL_PAROUV
       | UL_EGAL
       | UL_DIFF
       | UL_INF
       | UL_SUP
       | UL_ET
       | UL_OU
       | UL_SI
       | UL_ALORS
       | UL_FSI
       | UL_SINON
```

```
16        | UL_SUPEG
17        | UL_INFEG
18        | UL_IDENT of string
19
20 (** [is_eof] : token  -> bool
21     is_eof tk returns true if the lexical unit represents the end_of
          file.
22 *)
23 let is_eof = function
24   | UL_EOF -> true
25   | _      -> false
26
27 (** [print_token] : out_channel -> token -> unit
28     print_token o tk prints on the output channel o the textual
          representation of the token tk *)
29 let print_token o = function
30   | UL_EOF  -> Printf.fprintf o "UL_EOF"
31   | UL_PARFERM -> Printf.fprintf o "UL_PARFERM"
32   | UL_PAROUV -> Printf.fprintf o "UL_PAROUV"
33   | UL_EGAL -> Printf.fprintf o "UL_EGAL"
34   | UL_DIFF -> Printf.fprintf o "UL_DIFF"
35   | UL_INF -> Printf.fprintf o "UL_INF"
36   | UL_SUP -> Printf.fprintf o "UL_SUP"
37   | UL_ET -> Printf.fprintf o "UL_ET"
38   | UL_OU  -> Printf.fprintf o "UL_OU"
39   | UL_IDENT id  -> Printf.fprintf o "UL_IDENT \"%s\"" id
40   | UL_SI  -> Printf.fprintf o "UL_SI"
41   | UL_ALORS  -> Printf.fprintf o "UL_ALORS"
42   | UL_SINON  -> Printf.fprintf o "UL_SINON"
43   | UL_FSI  -> Printf.fprintf o "UL_FSI"
44   | UL_SUPEG  -> Printf.fprintf o "UL_SUPEG"
45   | UL_INFEG  -> Printf.fprintf o "UL_INFEG"
```

Listing 3: lexer.mll

```
1 {
2   open Ulex (* Ulex contains the type definition of lexical units *)
3 }
4
5 rule token = parse (* TODO *)
6    | [' ' '\t' '\n']     { token lexbuf } (* lexbuf : buffer *)
7    | "/*" ([^'*'] | '*'+ [^'/'])* '*'+'/' { token lexbuf } (*
        commentaires *)
8    | eof      { UL_EOF  }
9    | '('      { UL_PAROUV }
10   | ')'      { UL_PARFERM }
11   | '='      { UL_EGAL }
12   | "<>"     { UL_DIFF }
13   | '<'      { UL_INF  }
14   | '>'      { UL_SUP  }
15   | "et"     { UL_ET   }
16   | "ou"     { UL_OU   }
17   | "si"     { UL_SI   }
18   | "alors"     { UL_ALORS   }
19   | "sinon" { UL_SINON  }
20   | "fsi"    { UL_FSI   }
21   | ">="     { UL_SUPEG   }
22   | "<="     { UL_INFEG   }
```

```
23    | ['A'-'Z' 'a'-'z'] ['A'-'Z' 'a'-'z' '0'-'9' '_'] * as ident
        {UL_IDENT ident}
```

3 Tests

Listing 4: test.ml

```
1  let test1 = "jojo et /* commentaires */) cheval = sinon (";;
2  Printf.printf "TEST 1:\n";;
3  test (Lexing.from_string test1);;
4
5
6  let test2 = "si x=un alors y=deux sinon z=trois";;
7  Printf.printf "TEST 2:\n";;
8  test (Lexing.from_string test2);;
9
10 let test3 = "/** comm ** entaire/// ****/";;
11 Printf.printf "TEST 3:\n";;
12 test (Lexing.from_string test3);;
13
14 let test4 = "Le ciel est bleu /* NE PAS CONFONDRE AVEC LA MER */";;
15 Printf.printf "TEST 4:\n";;
16 test (Lexing.from_string test4);;
17
18 let test5 = "Mille millions de mille a";;
19 Printf.printf "TEST 5:\n";;
20 test (Lexing.from_string test5);;
21
22 let test6 = "si x=deux et y=trois /*abscisse ordonnee*/ alors z=un ou
       z=cinq sinon si z=deux alors echo test fsi fsi";;
23 Printf.printf "TEST 6:\n";;
24 test (Lexing.from_string test6);;
```

4 Résultats & Commentaires

```
1  TEST 1:
2    Token: UL_IDENT "jojo"
3    Token: UL_ET
4    Token: UL_PARFERM
5    Token: UL_IDENT "cheval"
6    Token: UL_EGAL
7    Token: UL_SINON
8    Token: UL_PAROUV
9    Token: UL_EOF
10   DONE
11   /*
12    Test du TP
13   */
14 TEST 2:
15   Token: UL_SI
16   Token: UL_IDENT "x"
17   Token: UL_EGAL
18   Token: UL_IDENT "un"
19   Token: UL_ALORS
20   Token: UL_IDENT "y"
```

```
  Token: UL_EGAL
  Token: UL_IDENT "deux"
  Token: UL_SINON
  Token: UL_IDENT "z"
  Token: UL_EGAL
  Token: UL_IDENT "trois"
  Token: UL_EOF
  DONE
  /*
  Test avec une condition.
  */
TEST 3:
  Token: UL_EOF
  DONE
  /*
  Aucune unite lexical car test avec seulement des commentaires
  */
TEST 4:
  Token: UL_IDENT "Le"
  Token: UL_IDENT "ciel"
  Token: UL_IDENT "est"
  Token: UL_IDENT "bleu"
  Token: UL_EOF
  DONE
  /*
  Test a avec seulement des idents
  */
TEST 5:
  Token: UL_IDENT "Mille"
  Token: UL_IDENT "millions"
  Token: UL_IDENT "de"
  Token: UL_IDENT "mille"
  Token: UL_IDENT "a"
  Token: UL_EOF
  DONE
  /*
  Test b avec seulement des idents
  */
TEST 6:
  Token: UL_SI
  Token: UL_IDENT "x"
  Token: UL_EGAL
  Token: UL_IDENT "deux"
  Token: UL_ET
  Token: UL_IDENT "y"
  Token: UL_EGAL
  Token: UL_IDENT "trois"
  Token: UL_ALORS
  Token: UL_IDENT "z"
  Token: UL_EGAL
  Token: UL_IDENT "un"
  Token: UL_OU
  Token: UL_IDENT "z"
  Token: UL_EGAL
  Token: UL_IDENT "cinq"
  Token: UL_SINON
  Token: UL_SI
  Token: UL_IDENT "z"
```

```
79  Token: UL_EGAL
80  Token: UL_IDENT "deux"
81  Token: UL_ALORS
82  Token: UL_IDENT "echo"
83  Token: UL_IDENT "test"
84  Token: UL_FSI
85  Token: UL_FSI
86  Token: UL_EOF
87  DONE
88  /*
89  Test avec des conditions imbriquees.
90  */
```

L'ensemble des tests ont fonctionné correctement vis à vis de la grammaire donnée du TP.