

TP1 - Analyse lexicale et crible

RAPHAËL BARON - PAUL CHAIGNON

24 septembre 2013

1 Analyseur

Listing 1 – ulex.ml

```
1 (** [token] is the type of the different lexical units. *)
2 type token = UL_IDENT of string
3   | UL_ET
4   | UL_OU
5   | UL_EGAL
6   | UL_EOF
7   | UL_PAROUV
8   | UL_PARFERM
9   | UL_DIFF
10  | UL_SUP
11  | UL_INF
12  | UL_SUPEGAL
13  | UL_INFEGAL
14
15 (** [is_eof] : token -> bool
16     is_eof tk returns true if the lexical unit represents the end_of
17     file.
18 *)
19 let is_eof = function
20   | UL_EOF -> true
21   | _      -> false
22
23 (** [print_token] : out_channel -> token -> unit
24     print_token o tk prints on the output channel o the textual
25     representation of the token tk *)
26 let print_token o = function
27   | UL_IDENT ident -> Printf.fprintf o "UL_IDENT %s" ident
28   | UL_ET          -> Printf.fprintf o "UL_ET"
29   | UL_OU          -> Printf.fprintf o "UL_OU"
30   | UL_EGAL        -> Printf.fprintf o "UL_EGAL"
31   | UL_PAROUV      -> Printf.fprintf o "UL_PAROUV"
32   | UL_PARFERM     -> Printf.fprintf o "UL_PARFERM"
33   | UL_DIFF        -> Printf.fprintf o "UL_DIFF"
34   | UL_SUP         -> Printf.fprintf o "UL_SUP"
35   | UL_INF         -> Printf.fprintf o "UL_INF"
36   | UL_SUPEGAL     -> Printf.fprintf o "UL_SUPEGAL"
37   | UL_INFEGAL     -> Printf.fprintf o "UL_INFEGAL"
38   | UL_EOF         -> Printf.fprintf o "UL_EOF"
```

Listing 2 – lexer.mll

```

1 {
2   open Ulex (* Ulex contains the type definition of lexical units *)
3 }
4 rule token = parse
5   | ([',', '\t', '\n'])+ {token lexbuf}
6   | "/*" ([^']*' | '*' + [^'/' '*'])* '*' + '/' {token lexbuf}
7   | "et" {UL_ET}
8   | "ou" {UL_OU}
9   | ['A'-'Z', 'a'-'z']+ as ident {UL_IDENT ident}
10  | ['1'-'9']+ as ident {UL_IDENT ident}
11  | ['=' ] {UL_EGAL}
12  | ['(' ] {UL_PAROUV}
13  | [')' ] {UL_PARFERM}
14  | "<>" {UL_DIFF}
15  | ['>' ] {UL_SUP}
16  | ['<' ] {UL_INF}
17  | ">=" {UL_SUPEGAL}
18  | "<=" {UL_INFEGAL}
19  | eof {UL_EOF}

```

2 Tests

Listing 3 – tests.ml

```

1 (* Test 1 *)
2 (* Expression simple, avec commentaire
3 (ouverture et fermeture avec une seule toile) *)
4 (*
5   /* Ceci est un test */ Test
6 *)
7 Token: UL_IDENT Test
8 Token: UL_EOF
9 DONE
10
11 (* Test 2 *)
12 (* Expression plus complexe (avec parenthse
13 et EGAL), ainsi que des toiles multiples en
14 ouverture/fermeture des commentaires *)
15 (*
16   /** Ceci est un autre test */
17   Si (essai = condition) alors test
18 *)
19 Token: UL_IDENT Si
20 Token: UL_PAROUV
21 Token: UL_IDENT essai
22 Token: UL_EGAL
23 Token: UL_IDENT condition
24 Token: UL_PARFERM
25 Token: UL_IDENT alors
26 Token: UL_IDENT test
27 Token: UL_EOF
28 DONE
29
30 (* Test 3 *)

```

```

31 (* Ingalits/diffrences/connecteurs
32 logiques, ainsi que de la reconnaissance des chiffres *)
33 (*
34   Si (1 <> 2) ou (3 > 4) alors 2 et 3
35 *)
36 Token: UL_IDENT Si
37 Token: UL_PAROUV
38 Token: UL_IDENT 1
39 Token: UL_DIFF
40 Token: UL_IDENT 2
41 Token: UL_PARFERM
42 Token: UL_OU
43 Token: UL_PAROUV
44 Token: UL_IDENT 3
45 Token: UL_SUP
46 Token: UL_IDENT 4
47 Token: UL_PARFERM
48 Token: UL_IDENT alors
49 Token: UL_IDENT 2
50 Token: UL_ET
51 Token: UL_IDENT 3
52 Token: UL_EOF
53 DONE

```

3 Questions

3.1 Question 1.1

Avoir un crible séparé permet de simplifier l'automate. Tout d'abord, le lexème va être reconnu par l'automate (valide ou non), puis grâce au crible on pourra lui attribuer l'unité lexicale qui correspond. Au final, la maintenance de code sera plus aisée.

3.2 Question 1.2

Oui, c'est parfaitement possible. Cela n'affectera pas l'AFD d'OCamllex, cependant, le crible sera lui modifié (afin d'associer correctement les unités lexicales et/ou).

3.3 Question 1.3

On peut distinguer 2 intérêts principaux : - On peut facilement effectuer un filtrage (objectif de l'analyse syntaxique), grâce aux "match .. with .." - L'ajout de nouveaux éléments reconnus est très simple : il suffit d'ajouter une ligne dans la définitions des types reconnus. Enfin, même si c'est plus anecdotique, cela permet d'avoir un code plus propre, et rapidement compréhensible.

3.4 Question 1.4

Grâce au scanner incrémental, on peut plus facilement voir où se trouver l'erreur, lorsqu'il y en a une. En effet, un scanner traditionnel permet simplement de savoir si l'expression est ou non reconnue, mais pas de situer l'erreur. A l'inverse, en incrémentant on sait exactement quel token pose problème.

3.5 Question 1.5

On considère `ident` comme un terminal, car ici on n'applique qu'une analyse lexicale, et non syntaxique. Un `ident` est un lexème, et est donc terminal.

3.6 Question 1.6

Il suffit de créer trois nouvelles unités lexicales : `UL_SI`, `UL_ALORS` et `UL_SINON`, et de les spécifier dans le crible.