TP1 - Analyse lexicale et crible

Raphaël Baron - Paul Chaignon

24 septembre 2013

1 Analyseur

Listing 1 – lexer.ml

```
(** [token] is the type of the different lexical units. *)
type token = UL_IDENT of string
       | UL_ET
       | UL_OU
       | UL_EGAL
            | UL_EOF
       | UL_PAROUV
       | UL_PARFERM
       | UL_DIFF
       | UL_SUP
       | UL_INF
       | UL_SUPEGAL
       | UL_INFEGAL

(** [is_eof] : token  -> bool
    is_eof tk returns true if the lexical unit represents the end_of
       file.
*)
let is_eof = function
  | UL_EOF -> true
  | _      -> false

(** [print_token] : out_channel -> token -> unit
    print_token o tk prints on the output channel o the textual
       representation of the token tk *)
let print_token o = function
  | UL_IDENT ident -> Printf.fprintf o "UL_IDENT %s" ident
  | UL_ET -> Printf.fprintf o "UL_ET"
  | UL_OU -> Printf.fprintf o "UL_OU"
  | UL_EGAL -> Printf.fprintf o "UL_EGAL"
  | UL_PAROUV -> Printf.fprintf o "UL_PAROUV"
  | UL_PARFERM -> Printf.fprintf o "UL_PARFERM"
  | UL_DIFF -> Printf.fprintf o "UL_DIFF"
  | UL_SUP -> Printf.fprintf o "UL_SUP"
  | UL_INF -> Printf.fprintf o "UL_INF"
  | UL_SUPEGAL -> Printf.fprintf o "UL_SUPEGAL"
  | UL_INFEGAL -> Printf.fprintf o "UL_INFEGAL"
  | UL_EOF  -> Printf.fprintf o "UL_EOF"
```

Listing 2 – lexer.mll

```
{
  open Ulex (* Ulex contains the type definition of lexical units *)
}
rule token = parse
  | ([' ' '\t' '\n'])+  {token lexbuf}
  | "/*" ([^'*'] | '*'+ [^'/''*'])* '*'* '/' {token lexbuf}
  | "et"  {UL_ET}
  | "ou"  {UL_OU}
  | ['A'-'Z' 'a'-'z']+ as ident {UL_IDENT ident}
  | ['1'- '9']+ as ident {UL_IDENT ident}
  | ['=']   {UL_EGAL}
  | ['(']   {UL_PAROUV}
  | [')']   {UL_PARFERM}
  | "<>"  {UL_DIFF}
  | ['>']   {UL_SUP}
  | ['<']   {UL_INF}
  | ">="  {UL_SUPEGAL}
  | "<="  {UL_INFEGAL}
  | eof     {UL_EOF}
  | "(*" {comments lexbuf}
```

2 Tests

Listing 3 – tests.ml

```
(* Test 1 *)
(* Expression simple, avec commentaire
(ouverture et fermeture avec une seule toile) *)
(*
  /* Ceci est un test */ Test
*)
Token: UL_IDENT Test
Token: UL_EOF
DONE

(* Test 2 *)
(* Expression plus complexe (avec parenthse
et EGAL), ainsi que des toiles multiples en
ouverture/fermeture des commentaires *)
(*
  /** Ceci est un autre test **/
  Si (essai = condition) alors test
*)
Token: UL_IDENT Si
Token: UL_PAROUV
Token: UL_IDENT essai
Token: UL_EGAL
Token: UL_IDENT condition
Token: UL_PARFERM
Token: UL_IDENT alors
Token: UL_IDENT test
Token: UL_EOF
DONE

```

```
(* Test 3 *)
(* Ingalits/diffrences/connecteurs
logiques, ainsi que de la reconnaissance des chiffres *)
(*
  Si (1 <> 2) ou (3 > 4) alors 2 et 3
*)
Token: UL_IDENT Si
Token: UL_PAROUV
Token: UL_IDENT 1
Token: UL_DIFF
Token: UL_IDENT 2
Token: UL_PARFERM
Token: UL_OU
Token: UL_PAROUV
Token: UL_IDENT 3
Token: UL_SUP
Token: UL_IDENT 4
Token: UL_PARFERM
Token: UL_IDENT alors
Token: UL_IDENT 2
Token: UL_ET
Token: UL_IDENT 3
Token: UL_EOF
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 pourrra lui attribuer l'unité lexicale qui correspond. Au final, la maintenace 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 simplemetn 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.