UFR d'Informatique
L3 Informatique, Math-Info et Japonais-Info

Grammaires et analyse syntaxique
Année 2023-2024

TP n°2

Ocamllex

Nous allons utiliser le générateur d'analyseur lexical `ocamllex`. On rappelle que `ocamllex` prend en entrée un fichier de spécification (dont le nom se termine par `.mll`) ayant trois/quatre parties :

- un prologue entre accolades `{` et `}` contenant du code Ocaml (typiquement des définitions utilisées dans la suite) qui sera placé au début du code engendré ;
- une séquence de définitions d'expressions rationnelles ;
- plusieurs points d'entrée de l'analyse lexicale qui regroupent une séquence d'expressions rationnelles avec les actions associées.
- un possible épilogue entre accolades `{` et `}` contenant du code OCaml qui sera placé à la fin du code engendré.

La façon la plus simple d'utiliser `ocamllex` est d'avoir un unique fichier `.mll` qui contient tout pour obtenir un programme exécutable. Prenons comme exemple le fichier `lexeur1.mll` qui contient un analyseur lexical qui prend un fichier en entrée, imprime tous les chiffres sur la sortie standard et ignore les autres caractères.

```
{
  exception Eof
  let liste = ref []
}

let digit = ['0'-'9']

rule lexeur = parse
  | digit as c  { liste := ((int_of_char c) - (int_of_char '0'))
     ::(!liste) }
  | _           { lexeur lexbuf }
  | eof   { raise Eof }

{
  let ch = open_in (Sys.argv.(1)) in
  let lexbuf = Lexing.from_channel ch in
  try
    while true do
      lexeur lexbuf
    done
  with Eof -> (List.iter print_int !liste; print_newline())
}
```

La génération d'un programme OCaml se fait avec `ocamllex lexeur1.mll`
la compilation ensuite avec `ocamlc -o main lexeur1.ml`
et l'exécution finalement avec `./main test.txt`

Exercice 1 *Examiner le programme* `lexeur1.ml` *engendré par ocamllex et repérer le prologue, l'épilogue, et la définition de* `digit`*.*

Exercice 2 *Modifier* `lexeur1.mll` *pour obtenir un programme qui calcule la somme de tous les chiffres d'un fichier.*

Une façon plus avancée d'utiliser ocamllex est de considérer les tokens. Pour cela on définit un type `token` dans un fichier à part (par exemple `token.ml`) qui contient les différents tokens que le lexeur peut utiliser et le lexeur produira une suite de tokens utilisée par exemple dans un programme principal `main.ml`. Par exemple, les trois fichiers `lexeur2.ml`, `token.ml` et `main.ml` ont la même fonctionnalité que l'unique fichier `lexeur1.ml`.

La compilation se fait soit laborieusement

```
ocamllex lexeur2.mll
ocamlc token.ml
ocamlc lexeur2.ml
ocamlc -o main lexeur2.cmo token.cmo main.ml
```

soit via un projet `dune` (fourni)

```
dune build main.exe
```

(l'exécution se fait dans ce cas avec `_build/default/main.exe test.txt`).

Exercice 3 *Sans modifier* `lexeur2.mll` *écrire un programme qui calcule le produit de tous les chiffres d'un fichier.*

Exercice 4 *Modifier les fichiers de sorte qu'on calcule la somme de tous les entiers (suite de chiffres) d'un fichier. Comment traiter les entiers négatifs (comme* `-111`*) ? Comment traiter les floats (comme* `11.11` *ou* `.11` *ou* `-.1` *ou* `13.`*) ?*
On n'hésitera pas à traiter de nouveaux types de jetons dans token.ml si besoin. Tester la solution obtenue sur le fichier `test.txt`*.*

Exercice 5 *Modifier les fichiers de sorte que pour chaque occurrence d'un entier le numéro de la ligne et la position sur la ligne soient indiqués. Voir la documentation de ocamllex.*

Pour aller plus loin : `https://github.com/ocaml/ocaml/blob/4.08/runtime/lexing.c`