Université de Paris – UFR d'Informatique
L3 Informatique et Math-Info

Grammaires et analyse syntaxique
Année 2020-2021

TP n°2

Ocamllex

Nous allons utiliser le générateur d'analyseur lexical `ocamllex`. On rappelle que `ocamllex` prend en entrée un fichier dont le nom se termine avec `.mll` qui a trois parties :

- un prologue entre accolades `{` et `}` contenant du code Ocaml (typiquement des définitions utilisées dans la suite) qui est placé au début du code OCaml généré ;
- 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.

Pour plus de détails voir le cours. On peut aussi ajouter un épilogue contenant du code OCaml à la fin entre accolades. Ce code est placé à la fin du code OCaml généré par `ocamllex`.
La façon la plus simple d'utiliser `ocamllex` est d'avoir un seul 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())
}
```

On génère un programme OCaml avec `ocamllex lexeur1.mll`
et on compile ensuite avec `ocamlc lexeur1.ml`.

Exercice 1 *Regarder le programme* `lexeur1.ml` *généré 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 `lexeur2.mll`, `token.ml` et `main.ml` ont la même fonctionnalité que `lexeur1.ml`. Pour compiler, on fait

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

ou mieux avec un fichier `Makefile`. Ceci produit un exécutable `main`.

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 *(facultatif) 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`