I53: Compilation et théorie des langages
TP 2

Semestre 2023-24

1 Traduction dirigée par la syntaxe

Le but de cette section est d'implanter les traducteurs vus en TD afin de se familiariser avec la traduction dirigée par la syntaxe.

1. Écrire un traducteur dirigé par la syntaxe transformant un nombre décimal en son équivalent en nombre romain.

```
> python nb2roman 4958
 MMMMCMDVIII
```

2. Écrire un traducteur dirigé par la syntaxe transformant une expression infixe en notation postfixe.

```
> python inf2post (4-3)*7/(2-1)
 4 3 - 7 * 2 1 - /
```

3. Écrire un traducteur dirigé par la syntaxe analysant les déclarations en C et produisant une description littérale de la variable déclarée . On ne prendra en compte que les types simples, les tableaux et les pointeurs. On pourra également ajouter le qualificateur `const` en omettant la construction `const <type>` pour simplifier l'analyse.

```
> python typeC "int * const tab[5]"
 tab est un tableau de 5 pointeurs constants sur int
```

2 Traducteur d'expression arithmétique vers machine RAM

Le but de cette partie est de réaliser un traducteur complet gérant tous les opérateurs précédents ainsi que des nombres entiers de taille arbitraire. On étendra ensuite le programme aux opérateurs de comparaisons ainsi qu'aux opérations booléennes afin d'obtenir un traduction complet. Pour cela le programme comportera quatre parties:

- un analyseur lexical
- un analyseur syntaxique chargé de transformer la chaine d'entrée en expression posfixée
- un producteur de code pour la machine RAM.

2.1 Analyseurs lexical

1. Écrire une fonction `lexer(s)` qui prend en paramètre une chaîne de caractère et retourne une liste d'unités lexicales. On considère pour cela le lexique suivant:

```
NOMBRE → 0|1|2|3|4|5|6|7|8|9
OP → + | - | * | /
PAR_OUV → (
PAR_FER → )
```

Par exemple pour l'entrée $5 - (9 + 2 * 3)$ la fonction retournera `[('NOMBRE',5),('OP','-'), ('PAR_OUV','('), (NOMBRE,9), ('OP', '+'), ('NOMBRE',2), ('OP','*'), ('NOMBRE', 3), ('PAR_FER',')')]`

2. Modifier l'analyseur pour que celui-ci ignore les blancs.

3. Étendre la définition de `NOMBRE` aux entiers de taille quelconque.

2.2 Analyseur syntaxique

1. Reprendre la grammaire des expressions arithmétiques vue en cours. Ajouter les opérations de soustraction, division ainsi que le moins unaire et nettoyer la grammaire afin de respecter l'associativité et la priorité des opérateurs et supprimer la récursivité à gauche.

2. Écrire une fonction `parser(ul)` qui prend en paramètre une suite d'unités lexical renvoyée par l'analyseur lexical et la traduit en notation postfixe à l'aide de la méthode de la traduction dirigée par la syntaxe.

2.3 Production de code

1. Écrire une fonction `codegen(ps)` qui prend en entrée une expression en notation postfixe et produit sous forme de fichier le code RAM prermettant son évaluation par la machine RAM.

2. Reprendre le programme complet pour ajouter la gestion des opérateurs de comparaison `<,>,=, !=`.

3 Analyseur d'expressions booléennes

Refaire le même travail avec la grammaire des expressions booléennes et intégrer la aux expressions arithmétiques afin de gérer les expressions arithmético-booléennes.