UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

Travaux Dirigés d'analyse syntaxique n°3
Licence d'informatique

———

Grammaires algébriques

L'objectif de ce TD est de prendre en main `bison`, d'apprendre à l'utiliser avec `flex`, et de se familiariser avec les grammaires algébriques en vue de l'analyse syntaxique.

▸ **Exercice 1.**

1. *Faire avec* `flex` *un analyseur lexical dans lequel chaque lettre est un lexème. Pour chaque lettre, l'analyseur doit renvoyer le caractère comme valeur de retour de* `yylex()`*. Il doit interpréter la fin de ligne comme lexème de fin de fichier, et ignorer les autres caractères. (On ne peut pas tester cet analyseur lexical avant d'avoir fait l'analyseur syntaxique.)*
2. *Faire avec* `bison` *un analyseur syntaxique pour la grammaire suivante et l'interfacer avec votre analyseur lexical :*
   $S \rightarrow aS \mid b$
3. *Faire un* `makefile` *pour les deux analyseurs.*
4. *Compiler et tester l'analyseur syntaxique sur quelques mots. Vérifier qu'il termine l'exécution tout de suite quand les mots sont reconnus par la grammaire, et qu'il produit un message d'erreur quand ils ne sont pas reconnus.*
5. *Ouvrir le programme en C engendré par* `bison` *et trouver l'appel de* `yylex()` *par* `yyparse()`*.*

▸ **Exercice 2.**

1. *Modifier le programme* `bison` *de l'exercice précédent pour que la valeur renvoyée par le* `main` *soit 1 quand le mot est reconnu par la grammaire, et 0 sinon. Tester votre analyseur sur un fichier.*
2. *Créer un script bash qui teste l'analyseur sur un dossier de fichiers de tests, et écrit les valeurs de retour dans un rapport de tests.*

▸ **Exercice 3.** *Pour chacune des grammaires suivantes, caractériser le langage engendré.*

1. $S \rightarrow aS \mid b$
2. $S \rightarrow Sa \mid b$
3. $S \rightarrow aaS \mid b$
4. $S \rightarrow aSa \mid b$
5. $S \rightarrow aSa$
6. $S \rightarrow aS \mid T$
   $T \rightarrow Ta \mid b$

▸ **Exercice 4.**

1. *Faire un analyseur lexical et un analyseur syntaxique pour la grammaire d'expressions booléennes suivante :*

$$\begin{array}{lcl} E_b & \rightarrow & E_b \textbf{ ou } T_b \mid T_b \\ T_b & \rightarrow & T_b \textbf{ et } F_b \mid F_b \\ F_b & \rightarrow & \textbf{non } F_b \mid (E_b) \mid \textbf{ vrai } \mid \textbf{ faux} \end{array}$$

   Dans l'analyseur lexical, les lexèmes sont **ou** , **et** , **non** , **vrai** , **faux** *et les parenthèses. La fin de ligne est le lexème de fin de fichier. Il doit pouvoir y avoir autant d'espace blanc qu'on veut entre deux lexèmes.*

2. *Adapter votre* `makefile` *si nécessaire : le programme en C engendré par* `flex` *doit être compilé après la mise à jour du fichier* `.h` *engendré par* `bison`.

3. *Compiler et tester sur quelques expressions booléennes.*

▸ **Exercice 5.**

1. *Caractériser le langage engendré par* $S \rightarrow aS \mid Sa \mid aaS \mid aSa \mid b$

2. *Adapter l'analyseur de l'exercice 1 à cette grammaire. Compiler, tester et expliquer ce qui se passe.*

▸ **Exercice 6.** *Proposer une grammaire utilisant un unique non-terminal pour décrire les expressions régulières sur l'alphabet* $\{a, b\}$ *sauf* ϵ. *Les opérateurs* $\cdot, +, *$ *et les parenthèses font partie de l'alphabet de la grammaire.*