UE 4TIN603U – Compilation – Licence 3 – 2020-2021

TD3 - Grammaire Algébrique

Les ressources pour débuter cette feuille de td se trouvent à l'URL suivante:

`https://moodle1.u-bordeaux.fr/mod/folder/view.php?id=365572`

1. **Analyser les expressions avec des automates ?**

   Un langage de programmation contient des *expressions* de toutes sortes, logiques, arithmétiques, de types, etc. Même l'écriture d'une structure de contrôle de type
   `if test then instruction else instruction` est une *expression*, car elle contient des opérateurs `if`, `then`, `else` et des instructions qui ne sont rien d'autre que des structures de contrôle.

   Il est aisé de se convaincre que toutes ces expressions reviennent à un mot bien parenthésé $a^n X b^n$ où a^n correspond à des parenthèses ouvrantes et b^n à des parenthèses fermantes, et X un contenu également bien parenthésé, de sorte que chaque parenthèse fermante b correspond à une parenthèse ouvrante a et qu'il y a autant de parenthèses fermantes que de parenthèses ouvrantes.

   Le plus simple des langages que l'on peut imaginer qui correspond au degré zéro d'une *expression* est le langage $\{a^n b^n / n \geq 0\}$ sur l'alphabet $\{a, b\}$.

   (a) Montrer que si l'on écrit un automate A qui contient exactement n états, tout chemin de longueur supérieure à n contient nécessairement au moins deux fois le même état. Intuitivement, que cela veut-il dire concernant le langage $\{a^n b^n / n \geq 0\}$?

   (b) Lemme de l'étoile (que nous admettons mais ne démontrons pas)

   Si L est un langage rationnel, alors il existe un entier N tel que tout mot ω du langage plus grand que N se décompose en $\omega = xyz$ et tel que :
   — $|xy| \leq N$, $|y| \neq 0$;
   — pour tout entier $i > 0$, $xy^i z \in L$.

   On va montrer, en utilisant le lemme de l'étoile, que si l'on suppose que $\{a^n b^n / n \geq 0\}$ est rationnel, alors on aboutit à une contradiction.

   En effet, si le langage est rationnel, alors d'après le lemme, il existe un entier N tel que la propriété du lemme soit vraie. Considérons alors le mot $\omega = a^N b^N$, qui est par construction plus grand que N (il fait deux fois N).

   D'après le lemme, il doit se décomposer en $\omega = xyz$ tel que : $|xy| \leq N$ et $|y| \neq 0$.
   — Que peut-on en conclure sur la forme de x, y et z ?
   — Où se trouve alors la contradiction ?

2. **Grammaire $a^n b^n$ – Premier programme avec `Bison`**

   Nous commençons par reconnaitre la grammaire $a^n b^n$ à l'aide d'un analyseur syntaxique.

   Documentation de `Bison` : `http://www.gnu.org/software/bison/manual/`

   — Créer un nouveau projet `td3.2`
   — Télécharger `Parser.y` `build.xml` et `Makefile` et construire un nouveau projet avec son contenu. `Parser.y` devra se trouver dans un répertoire `parser`.
   — Lancer la commande `ant parser` ou `make parser`.
   Cette action va produire :
   — `src/Parser.java`
   — Consulter le fichier produit, puis compiler avec la commande `ant bin` ou `make bin`.
   Cette action va produire les fichiers `bin/*.class`.
   — Lancer le programme avec la commande `ant` ou `make`.
   Vous devrez rentrer une chaine de caractères et terminer par un retour à la ligne (newline).
   Astuce : dans une console Unix, taper `^d` permet d'arrêter la saisie sans produire le code `\n`.
   — Modifier `Parser.y` pour que le programme lise un un mot ω avec $\omega \in \{a^n b^n / n \geq 0\}$
   — Pour observer le comportement de l'analyseur, modifier `Parser.y` pour que le programme affiche le nom de la règle pour chaque règle réduite.

3. **Interface avec JFlex**

   Dans ce qui précède, l'analyse lexicale a été directement fournie à la fin de `Parser.y` dans une classe `MyLexer` qui implémente `Parser.Lexer`.

   (a) Consulter l'interface `Parser.Lexer` produite par *Bison* pour y découvrir ses méthodes. Lire les commentaires qui indiquent la sémantique de ces méthodes pour bien les comprendre.

   (b) Construire un analyseur syntaxique avec JFlex qui permet d'implémenter l'interface `Parser.Lexer`.

   Remarque : `Lexer.jflex` devra se trouver dans le répertoire `lexer` et il faudra modifier `build.xml` (ou `Makefile`).

4. **Grammaire des expressions de la logique propositionnelle**

   Laissons de côté le langage $a^n b^n$ qui était juste un exemple simpliste, et passons au langage L des expressions de la logique propositionnelle, qui se définit ainsi :

   — Constantes
   — $1 \in L$ (vrai)
   — $0 \in L$ (faux)
   — Variables
   — $p \in L$ où p est le nom d'une variable propositionnelle
   — Formule unaire Si E est une expression de la logique propositionnelle
   — $\neg\, E \in L$
   — Formules binaires Si E et F sont des expressions de la logique propositionnelle
   — $(E \vee F) \in L$ (ou)
   — $(E \wedge F) \in L$ (et)

(a) Reprendre le projet de la section précédente, supprimer les règles ajoutées précédemment et l'analyseur lexical.

(b) Écrire un analyseur lexical en `JFlex` qui reconnaît l'ensemble des constantes, les variables, les opérateurs et les parenthèses du langage des expressions de la logique propositionnelle, tel que décrit plus haut.

Encodage Unicode hexadécimal	Caractère
`\u2228`	$\vee$
`\u2227`	$\wedge$
`\u00AC`	$\neg$

(c) On remarquera que l'écriture courante des expressions se passe parfois de parenthèses dans la mesure où nous savons que $\wedge$ est prioritaire sur $\vee$. Ainsi, $(p \vee (q \wedge r))$ pourra aussi s'écrire $p \vee q \wedge r$. Pour simplifier, nous pourrons dire qu'une expression est une disjonction de plusieurs termes ou un terme unique, et que chaque terme est une conjonction de plusieurs facteurs ou un facteur unique. Enfin un facteur est une variable propositionnelle, une constante ou encore une expression entière notée entre parenthèses ou la négation d'un facteur.

Écrire un analyseur syntaxique selon les spécifications données ci-dessus.

(d) Ajouter les numéros de lignes et de colonnes aux symboles de manière à ce qu'un affichage de ces informations soit réalisé en cas d'erreur de tokenisation ou de syntaxe.

(e) Ajouter du code à chaque règle de la grammaire pour afficher le nom de la règle réduite. Qu'observe-t-on ?