Contrôle continu - Compilation (L3 Info)
Durée : 2h - documents interdits

Aix Montperrin - 23 février 2016

Conventions

— Axiome : symbole non terminal à gauche de la première production de la grammaire
— Symboles non terminaux : lettres *MAJUSCULES ITALIQUES*
— Symboles terminaux : lettres `minuscules true-type` ou caractères spéciaux simples

Exercice 1 - Écriture de grammaire *(7 pts)*
En Python, on peut déclarer des fonctions "anonymes" en utilisant la notation *lambda*. Par exemple, l'affectation ci-dessous déclare une lambda-fonction et l'affecte à la variable `f`. Ensuite, la fonction est évaluée et le résultat est affiché.

```
>>> f = lambda x, y: x + y
>>> print f(3,2)
5
```

La déclaration et l'évaluation de lambda-fonctions sont des *expressions* du langage Python. Nous définissions un sous-ensemble des expressions Python comprenant :

— Une *expression* du type déclaration de lambda-fonction, composée de :
  1. le mot-clef `lambda`, puis
  2. une liste potentiellement vide d'*identificateurs* séparés par virgules (arguments de la fonction), puis
  3. deux-points (`:`), puis
  4. une *expression* (définition/corps de la fonction).
— Une *expression* du type évaluation de lambda-fonction, composée de :
  1. un identificateur, puis
  2. une liste non-vide de paires de parenthèses ouvrantes et fermantes équilibrées et non-imbriquées. À l'intérieur des parenthèses, il y a une liste potentiellement vide d'*expressions* correspondant aux valeurs des arguments.
— Une *expression* du type arithmétique peut être :
  — un nombre seul (`nb`), ou
  — un identificateur seul (`id`), ou
  — une somme d'expressions (`+`),
  — une multiplication d'expressions (`*`).

La multiplication est prioritaire par rapport à la somme. L'évaluation d'une lambda-fonction a la plus haute précédence au même niveau que les nombres et les identificateurs. La déclaration de lambda-fonction a la plus basse précédence avec la somme. Une lambda-fonction peut avoir une liste d'arguments vide. La déclaration de lambda-fonction `f = lambda : 1` est acceptable. Son évaluation prend la forme `f()`. Une lambda-fonction contient une expression. Elle peut donc contenir une autre lambda-fonction, comme dans l'exemple ci-dessous :

```
>>> f = lambda x: lambda y: x + y
>>> print f(3)(2)
5
```

Pour répondre aux questions ci-dessous, supposez qu'un analyseur lexical renvoie les symboles terminaux `id` pour un identificateur (de variable ou de fonction), `nb` pour un nombre, les symboles d'un seul caractères `, : + * ( )` et le mot-clef `lambda`. Vous pouvez répondre aux questions 1-3 avec une seule grammaire. Votre grammaire doit être non-ambiguë mais elle peut ne pas être $LL(1)$.

Écrivez une grammaire non-ambiguë qui permet d'engendrer le sous-langage d'expressions Python du type :

1. déclaration de lambda-fonction
2. évaluation de lambda-fonction
3. arithmétique
4. Dessinez l'arbre de dérivation du mot `lambda id : nb`

Exercice 2 - Analyse LL(1) et ambiguïté *5 pts*

La grammaire G_{if} ci-dessous décrit la construction `if-else` en C. Pour simplifier, nous supposerons que `pe` est un noeud terminal qui correspond à une expression analysée entre parenthèses et `ins` est un terminal qui correspond à une instruction quelconque autre que `if-else` et bloc entre accolades. Le seul non-terminal de cette grammaire est I.

$$
\begin{array}{lcl}
I & \rightarrow & \texttt{if pe } I \\
I & \rightarrow & \texttt{if pe } I \texttt{ else } I \\
I & \rightarrow & \texttt{\{ } I \texttt{ \}} \\
I & \rightarrow & \texttt{ins ; } I \\
I & \rightarrow & \texttt{ins ;}
\end{array}
$$

1. Réécrivez la grammaire en la factorisant à gauche pour obtenir G'_{if}
2. Calculez les ensembles Premier et Suivant et construisez la table LL(1) de G'_{if}
3. Démontrez que la grammaire G_{if} OU la grammaire G'_{if} est ambiguë
4. Quelle est la solution adoptée en L pour rendre l'instruction `si-sinon` non ambiguë ?

Exercice 3 - Analyse $LL(1)$ *6 pts*

Soit la grammaire G_1 :

$$
\begin{array}{lcl}
S_1 & \rightarrow & S_2\ S_3 \\
S_2 & \rightarrow & \texttt{a}\ S_2 \mid \varepsilon \\
S_3 & \rightarrow & \texttt{b}\ S_3 \mid \texttt{c}
\end{array}
$$

1. Décrivez en 1-2 phrases ou en notation mathématique le langage engendré par G_1
2. Calculez les ensembles Premier et Suivant de G_1
3. Construisez la table LL(1) de G_1
4. Simulez l'analyse $LL(1)$ du mot `abc`. Les configurations sont représentées par le triplet (w, α, y), où w est la partie de la bande de lecture qui commence au caractère sous la tête de lecture (ce qui reste à analyser), α est la pile et y est la séquence de symboles de sortie (numéros des productions appliquées).

Exercice 4 - Récursivité *2 pts*

Soit la grammaire G_2 :

$$
\begin{array}{lcl}
S & \rightarrow & A\ \texttt{a} \mid \texttt{a} \\
A & \rightarrow & B\ \texttt{c} \\
B & \rightarrow & S\ \texttt{b} \mid B\ \texttt{b} \mid \varepsilon
\end{array}
$$

1. Donnez une nouvelle version de G_2 sans récursivité gauche directe ni indirecte.