Compilation : Langages et Grammaires

EPITA – Promo 2006 – **Tous documents autorisés** *

Janvier 2004 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur.

Les résultats ne doivent pas être balancés comme « évidents », sous peine de disqualification pour tentative de bluffage. Néanmoins, une argumentation informelle mais convaincante, sera souvent suffisante.

Dans cette épreuve, les non-terminaux sont écrits en majuscules, les terminaux en minuscules.

1 Hiérarchie de Chomsky

Pour chacun des langages suivants,
- préciser son type dans la hiérarchie de Chomsky :
  - son rang, e.g., 3
  - son nom, e.g., langage régulier

  On demande **évidemment** le type le plus précis.
- proposer une grammaire
  - en syntaxe BNF
  - qui l'engendre
  - qui ait le même type de Chomsky
  - qui soit non ambiguë.

  Les grammaires longues peuvent être esquissées et se terminer par « ... ».

1. Les logins de la promo 2006 de l'EPITA.
2. Les nombres entiers non signés écrits en base 4 (i.e., composés de « 0 », « 1 », « 2 », « 3 »).
3. Les sommes (« + ») de nombres binaires.
4. Les soustractions de nombres binaires avec parenthèses (« (», «) »).

*"Tout document autorisé" signifie que notes de cours, livres, annales, etc. sont explicitement consultables pendant l'épreuve. Le zèle de la part des surveillants n'a pas lieu d'être, mais dans un tel cas contacter le LRDE au 01 53 14 59 22.

2 Parsage LL(1)

Lorsque l'on s'attaque à des problèmes sur des structures évoluées (telles que des arbres), on constate souvent que le code mélange à la fois le traitement proprement dit (par exemple changer l'étiquette d'un nœud), et le parcours (appliquer à tous les fils, ou bien encore essayer tel traitement, puis en cas de succès, en appliquer tel autre etc.).

Une façon puissante d'exprimer de tels programmes consiste en la séparation des traitements et des parcours. Ces modes de parcours sont appelés *stratégies*. On s'intéressera alors à un langage dédié aux stratégies, i.e., permettant de composer des parcours. Stratego est un exemple de tel langage, dédié à la transformation de programmes (`http://www.stratego-language.org/twiki/bin/view/Stratego`).

Les stratégies de base comptent la stratégie `0` (échec), la stratégie `1` (succès), ou encore `s` qui dénote une quelconque stratégie atomique de l'utilisateur. À partir de deux stratégies s_1 et s_2 peuvent être construites les stratégies s_1 `+` s_2 (choix non déterministe), s_1 `<+` s_2 (choix gauche), s_1 `.` s_2 (composition séquentielle). Les parenthèses permettent de grouper.

Ce langage inclut des mots tels que :

```
0 <+ 1        (s + s) . 1        s . s <+ s <+ 1
```

1. Écrire une grammaire hors contexte naïve de ce langage de stratégies. On cherchera une formulation abstraite, courte et très lisible, au prix de l'ambiguïté.
2. Désambiguïser cette grammaire en considérant les règles suivantes :
   (a) Toutes les opérations binaires sont associatives à gauche ;
   (b) `.` est prioritaire sur `<+` ;
   (c) `<+` est prioritaire sur `+`.

   c'est-à-dire que

   ```
   0 + s . s <+ s <+ 1
   ```

   se lit comme suit.

   ```
   0 + (((s . s) <+ s) <+ 1)
   ```

3. Expliquer pourquoi cette grammaire ne peut pas être LL(1).
4. Transformer cette grammaire en une grammaire susceptible d'être LL(1).
5. Quelle critique formuler sur la grammaire obtenue ?
6. Récrire cette grammaire en s'autorisant les extensions de l'EBNF. Par exemple,

   ```
   A ::= a*.
   ```

7. Écrire en pseudo code un parseur LL(1) avec les bonnes priorités et associativités pour cette grammaire.
   Il suffira d'écrire une et une seule des routines de parsage, à condition qu'elle soit significative (comprendre que `eat`, aussi appelée `accept`, n'est pas demandée). On prendra soin de ne pas cacher la gestion des erreurs.