Correction du Partiel de Rattrapage Compilation : Langages et Grammaires

EPITA – Promo 2006

Avril 2004 (30mn)

Aucun document autorisé[1]

Soit un langage de *stratégies*. 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.

Correction:

```
S ::= S .  S
    | S <+ S
    | S +  S
    | ( S )
    | 0 | 1 | s
```

Barème:

– Grammaire correcte : 4
– Étourderie, mais grammaire naïve et simple : 2

Best-of:

```
– S -> E $
  E -> T | ϵ
  T -> C <+ C | (C + C) | T . 1 | T <+ T | C | ϵ
  C -> 0 | 1 | ϵ
– S -> "0" | "1" | "s"
  T -> "(" | ")" | "." | "<+"
  P -> S P1
  P1 -> T S P1 | ϵ
– S => C A C
  A => 0|1|s
  B => <+ | .
  C => ϵ | ( | )
  A => C A B C
  B => C B A C
```

1. « Aucun document autorisé » signifie que ni notes de cours, livres, annales, etc. ne sont consultables pendant l'épreuve.

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)
   ```

Correction: Comme pour l'arithmétique, on introduit des symboles non-terminaux supplémentaires pour chaque étage de priorité. On prendra `S` (somme) pour l'étage `+`, `C` pour le choix `<+`, `T` (terme) pour l'étage `.`, et `F` (facteur) pour les valeurs littérales et les parenthèses.

```
// Une somme est une somme de choix ou un choix.
S ::= S + C
S ::= C
// Un choix est un choix parmi des termes, ou un terme.
C ::= C <+ T
C ::= T
// Un terme est un produit de facteurs, ou un facteur.
T ::= T . F
T ::= F
// Un facteur est un littéral.
F ::= 0 | 1 | s | ( S )
```

On remarque la récursivité à gauche des règles pour obtenir l'associativité à gauche.

Barème:
– Bonne désambiguïsation : 4
– Priorités strictement inversées : 2

Best-of:
- pas le temps.
  récursion à droite + factorisation à gauche.
- (= somme) pour +
  T (Terme) pour <+
  F (Fuckem) pour .
  et E pour stratégie (pourquoi pas ?)
- (a) `S -> S + E | E`
  `E -> E . F | F`
  `F -> F <+ G | G`
  `G -> 0 | 1 | s (S)`
  (b) `S -> S <+ E | E`
  `E -> E + F | F`
  `F -> F . G | G`
  `G -> 0 | 1 | s (S)`
  (c) `S -> S + E | E`
  `E -> E <+ F | F`
  `F -> F . G | G`
  `G -> 0 | 1 | s (S)`
- Bon, alors là, je ne trouve pas le sujet clair. Est-ce que si je n'explique pas comment j'arrive à cette grammaire, je "bluffe" ? La méthode est dans les 4 supports que j'ai emmené (notes de cours, th-lang.pdf, Appel, [? ? ?]), on l'a fait et refait en cours... bon après bien sur j'ai l'air con si j'ai fait une erreur.

3. Expliquer pourquoi cette grammaire ne peut pas être LL(1).

Correction: D'une part elle est récursive à gauche, d'autre part, plusieurs règles sont en concurrence. Par exemple les deux premières règles (celles de S) sont actives pour tout FIRST de C.

Barème:
- Récursion : +2
- Concurence : +2

Best-of:
- Cette grammaire ne peut pas être LL(1) à cause des priorités des opérateurs qui nécessitent d'aller voir plus loin.
- C'est similaire à la grammaire du partiel de l'année dernière, "les deux premières règles sont actives pour tout FIRST de T".
- LL(1) signifie que l'on part systématiquement de l'élément le plus à gauche, et qu'on ne regarde que le token suivant, rien de plus. On ne peut donc pas (avec la grammaire LL(1)) écrire une grammaire comme celle de l'exercice (logique).
- Cette grammaire ne peut pas être LL1 car elle possède la régularité à gauche. elle n'est donc pas régulière.
- Cette grammaire ne peut pas être LL(1) car d'après le cours, toute grammaire LL(1) est non ambiguë, comme celle-ci est ambiguë, elle n'est pas LL(1).
- Elle est récursive à droite.
- LL ne sait pas gérer les récursions gauche.
  $\Rightarrow$ on dérécursionne (ré-récursionne ?) à droite.
- Cette grammaire n'est pas LL car pour des grammaires tels que U on a trop de productions possibles.

4. Transformer cette grammaire en une grammaire susceptible d'être LL(1).

Correction:

(a) Récursion à droite.

```
S ::= C + S  | C
C ::= T <+ C | T
T ::= F . T  | F
F ::= 0 | 1 | s | ( S )
```

(b) Factorisation à gauche En utilisant ” pour désigner le mot vide.

```
S  ::= C S'
S' ::= + S  | ''
C  ::= T C'
C' ::= <+ C | ''
T  ::= F T'
T' ::= . T  | ''
F  ::= 0 | 1 | s | ( S )
```

(c) Simplification des récursions mutuelles.

```
S  ::= C S'
S' ::= + C S'  | ''
C  ::= T C'
C' ::= <+ T C' | ''
T  ::= F T'
T' ::= . F T'  | ''
F  ::= 0 | 1 | s | ( S )
```

Barème:
– Récursion et factorisation : 4
– Récursion seule (mais concorde avec la réponse 3) : 2

Best-of:
– On va donc mettre la récursion à droite :

```
S -> E $
E -> I "+" E | I
I -> T "<+" I | T
T -> F "." T | F
F -> 0 | 1 | s | (E)
```

(Note : un doute subsiste à savoir si "." doit appartenir à l'etage des parenthèses, ou bien rester à son étage).
Ainsi il aurait peut etre suffit d'écrire :

```
S -> E $
E -> T + E | T
T-> F <+ T F
F ->  0 | 1 | s | (E) | E.F
```

(ce qui me gène ici, c'est le fait que "." soit binaire et non unaire).
– On force donc l'utilisation des parenthèses :

```
S -> S + (E) | E
E -> E <+ (F) | F
F -> F . (G) | G
G -> (S) | L
L -> s | 0 | 1
```

5. Quelle critique formuler sur la grammaire obtenue ?

Correction: On a perdu l'associativité gauche des opérateurs.

Barème:
– Bonne réponse- : 4 points.
– Mauvaise réponse- : 0 point.

Best-of:
– C'est illisible et totalement contre-intuitif.
– La grammaire obtenue engendre des langages trop parenthésés, ce qui est lourd et contraignant. On se croirait en LISP.

6. Récrire cette grammaire en s'autorisant les extensions de l'EBNF. Par exemple,

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

Correction:

```
S ::= C (+ C)*
C ::= T (<+ T)*
T ::= F (. F)*
F ::= 0 | 1 | s | ( S )
```

Barème:
– Grammaire correcte- : 4 points.
– Mauvaise priorité- : 3 points.
– Pour chaque faute dans la grammaire- : -1 point.

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.

Correction: `S` se parse simplement par une routine :

```
routine parse-S ()
  parse-C ()
  tant que lookahead égale '+' faire
    accepter '+'
    parse-C ()
  fin tant que
fin routine parse-S
```

et de façon similaire pour `C` et `T`. Enfin, toutes les règles de `F` commencent par un terminal différent donc aucun problème pour LL.

Barème:
– Code correct : 4 points.
– Erreur d'étourderie (mauvais opérateur) : -1 point.