Analyse syntaxique automatique

Parsing top-down et parsing bottom-up

Solutions du devoir du 7.3.2001

1. (a) Problèmes pour le parsing top-down et le parsing bottom-up :
   top-down : récursion à gauche (NP $\rightarrow$ NP Srel)
   bottom-up : production avec mot vide comme partie droite (NP-e $\rightarrow \epsilon$)

   (b) Transformer G en grammaire faiblement équivalente qui permet un parsing top-down :
   Choisir un nouveau symbole non-terminal NP' et l'ajouter à la grammaire.
   Remplacer les productions
   NP $\rightarrow$ det n   NP $\rightarrow$ n   NP $\rightarrow$ NP Srel
   par
   NP $\rightarrow$ det n   NP $\rightarrow$ n
   NP $\rightarrow$ det n NP'   NP $\rightarrow$ n NP'
   NP' $\rightarrow$ Srel   NP' $\rightarrow$ Srel NP'
   ou alternativement par
   NP $\rightarrow$ det n NP'   NP $\rightarrow$ n NP'
   NP' $\rightarrow$ Srel NP'   NP' $\rightarrow \epsilon$

   (c) Transformer G en grammaire faiblement équivalente qui permet un parsing bottom-up :
   Remplacer VP-e $\rightarrow$ v NP-e par NP-e $\rightarrow$ v et supprimer NP-e $\rightarrow \epsilon$

2. (a) 4 6 9 0 10 0 6 8 0 7 (0)
   L'algorithme au cours ne va pas mettre le dernier 0, mais on peut très bien avoir une variante qui enregistre le premier shift en mettant d'abord un 0.

   (b) Obtenir une analyse à partir de *stack*3 :

Algorithme structure-bu

```
Données globales :  un dictionnaire D et une grammaire G = ⟨V,T,S,P⟩
Input :  une pile stack3
Output :  une structure
Méthode :
   if stack3 = []
   then return([])
   else return(structure-help())
```

Procédure structure-help

```
Données globales :  dictionnaire D, grammaire G = ⟨V,T,S,P⟩ et pile stack3
Input :  -
Output :  une structure
Effet supplémentaire :  modification de stack3
Variable locale :  index m, structure res
Méthode :
   m := top(stack3)
   stack3 := pop(stack3)
   if D_m = ⟨mot, x⟩ ∈ D
   then
      if stack3 ≠ []
      then stack3 := pop(stack3)
      return([x mot])
   else if P_m ∈ P
      then
         res := []
         for i = 1, ..., |right-side|
            res := [structure-help()] + res
         return([left(P_m)] + res)
```

(c) Analyse de *le joli chat rouge* : trace des appels récursifs de `structure-help`. Valeur initiale de $stack3$: [4 6 9 0 10 ...]. $stack3$ est une variable globale, une variable locale var avec un index i, $var(i)$ indique qu'il s'agit de la variable locale de l'appel i de `structure-help`.

$stack3 =$ [4 6 9 0 10 0 6 8 0 7 0]

    appel 1 :
    m(1) = 4
    stack3 = [6 9 0 10 0 6 8 0 7 0]
    res(1) = [ ]
        appel 2 :
        m(2) = 6
        stack3 = [9 0 10 0 6 8 0 7 0]
        res(2) = [ ]
            appel 3 :
            m(3) = 9
            stack3 = [10 0 6 8 0 7 0]
            appel 3 retourne [a rouge]
        res(2) = [[a rouge]]
        appel 2 retourne [AP [a rouge]]
    res(1) = [[AP [a rouge]]]
        appel 4 :
        m(4) = 10
        stack3 = [6 8 0 7 0]
        appel 4 retourne [n chat]
    res(1) = [[n chat][AP [a rouge]]]
        appel 5 :
        m(5) = 6
        stack3 = [8 0 7 0]
        res(5) = [ ]
            appel 6 :
            m(6) = 8
            stack3 = [7 0]
            appel 6 retourne [a joli]
        res(5) = [[a joli]]
        appel 5 retourne [AP [a joli]]
    res(1) = [[AP [a joli]][n chat][AP [a rouge]]]
        appel 7 :
        m(7) = 7
        stack3 = [ ]
        appel 7 retourne [det le]
    res(1) = [[det le][AP [a joli]][n chat][AP [a rouge]]]
    appel 1 retourne [NP [det le] [AP [a joli]] [n chat] [AP [a rouge]]]