Université de Nice-Sophia Antipolis — Licence d'Informatique
Module d'analyses — 2004–2005

Examen de contrôle continu blanc

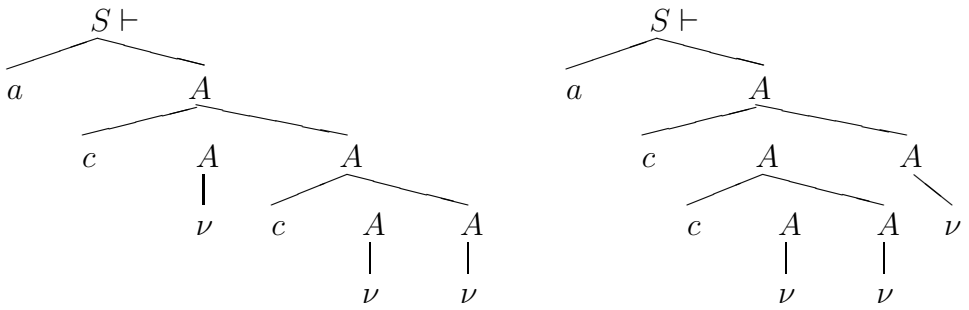
Durée : 2 heures
Tous documents autorisés

1 Arbre de dérivation

Grammaire G_1

Axiome = S
$N_1 = \{S, A\}$
$T_1 = \{a, c\}$
$P_1 = \{\ S \to aA \quad \}$
$S \to a$
$A \to cAA$
$A \to \nu$

a. Dessinez deux arbres de dérivation dans G_1 pour la chaîne terminale `acc`. Que peut-on en déduire sur G_1 ?



La grammaire G_1 a deux dérivations différentes pour une seule chaîne terminale ; elle est ambigüe.

. . . . . .

2 Grammaire fortement LL(k)

On rappelle ici la caractérisation des grammaires fortement LL(k) :

Soit G = <S, N, T, P> une grammaire algébrique. La grammaire G est fortement LL(k) pour $k > 0$ si et seulement si, pour tout couple de productions $A \to \alpha$ et $A \to \beta$ avec $\alpha \neq \beta$,

$$\text{PREM}_k(\alpha\ \text{SUIV}_k(A)) \cap \text{PREM}_k(\beta\ \text{SUIV}_k(A)) = \emptyset.$$

Grammaire G_2

Axiome = S

$N_2 = \{S, A\}$

$T_2 = \{a, b\}$

$P_2 = \{ \; S \to bAbb \;\; \}$

$S \to aAab$

$A \to a$

$A \to \nu$

b. Montrez que cette grammaire n'est pas fortement LL(2), mais qu'elle est fortement LL(3).

On applique la formule de caractérisation des grammaires fortement LL(k) aux productions $A \to a$ et $A \to \nu$:

$$\text{PREM}_2(a \; \text{SUIV}_2(A)) = \{ab, aa\} \tag{1}$$

et

$$\text{PREM}_2(\nu \; \text{SUIV}_2(A)) = \{bb, ab\}. \tag{2}$$

L'intersection des ensembles (1) et (2) n'est pas vide, donc G_2 n'est pas fortement LL(2).

En revanche, pour $k = 3$, on obtient :

$$\text{PREM}_3(bAbb \; \text{SUIV}_3(S)) = \{bab, bbb\}, \tag{3}$$

$$\text{PREM}_3(aAab \; \text{SUIV}_3(S)) = \{aaa, aab\}, \tag{4}$$

$$\text{PREM}_3(a \; \text{SUIV}_3(A)) = \{abb, aab\} \tag{5}$$

et

$$\text{PREM}_3(\nu \; \text{SUIV}_3(A)) = \{bb\vdash, ab\vdash\}. \tag{6}$$

Ces ensembles sont bien disjoints. □

. . . . . .

3 Caractérisation LL(k)

Voici une caractérisation des grammaires LL(k) assez semblable à la caractérisation que vous connaissez pour les grammaires fortement LL(k) :

Soit une grammaire algébrique $G = <S, N, T, P>$. La grammaire G est LL(k) pour $k > 0$ si et seulement si, pour tout couple de productions $A \to \alpha$ et $A \to \beta$ avec $\alpha \neq \beta$, et pour toute chaîne σ de V^* avec $S \Rightarrow^G xA\sigma$ pour un certain x de T^* :

$$\text{PREM}_k(\alpha\sigma) \cap \text{PREM}_k(\beta\sigma) = \emptyset.$$

c. Vous allez appliquer cette caractérisation pour montrer que G_2 est LL(2). Pour cela vous allez considérer successivement les chaînes bb et ab en guise de σ.

En prenant bb en guise de σ, les ensembles

$$\text{PREM}_2(abb) = \{ab\} \tag{7}$$

et

$$\text{PREM}_2(\nu bb) = \{bb\} \tag{8}$$

d'une part, et en prenant ab en guise de σ, les ensembles

$$\text{PREM}_2(aab) = \{aa\} \tag{9}$$

et

$$\text{PREM}_2(\nu ab) = \{ab\} \tag{10}$$

d'autre part sont bien disjoints. Or, on a bien les dérivations $S \Rightarrow^G bAbb$ et $S \Rightarrow^G aAab$, donc la caractérisation LL(2) est vérifiée pour les productions de A.

Par ailleurs la caractérisation est trivialement vérifiée pour les productions de S. La grammaire G_2 est donc LL(2). □

. . . . . .

d. Montrez en utilisant cette caractérisation que toute grammaire fortement LL(k) est LL(k).

Soit G = <S, N, T, P> une grammaire fortement LL(k) : pour tout couple de productions $A \to \alpha$ et $A \to \beta$ avec $\alpha \neq \beta$,

$$\text{PREM}_k(\alpha\ \text{SUIV}_k(A)) \cap \text{PREM}_k(\beta\ \text{SUIV}_k(A)) = \emptyset. \tag{11}$$

Rappelons que

$$\text{SUIV}_k(A) = \{\tau \mid \exists \phi \in V^* \text{ tel que } S \Rightarrow^* \mu A \phi, \tau \in \text{PREM}_k(\phi)\}; \tag{12}$$

et donc, si une chaîne σ est telle que $S \Rightarrow^G xA\sigma$ pour un certain x de T^*, alors

$$\text{PREM}_k(\sigma) \subseteq \text{SUIV}_k(A). \tag{13}$$

Dès lors, les équations (11) et (13) nous assurent que σ vérifie bien

$$\text{PREM}_k(\alpha\sigma) \cap \text{PREM}_k(\beta\sigma) = \emptyset, \tag{14}$$

et que la grammaire G est LL(k). □

. . . . . .

4 Analyseur syntaxique en descente récursive

e. Écrivez un programme C qui vérifie qu'une chaîne terminale fasse bien partie de $\mathcal{L}(G_2)$ et qui écrit sur la sortie standard le nombre de a et de b contenus dans la chaîne. Le vocabulaire étant réduit à l'alphabet $\{a, b\}$, vous fournirez l'analyseur lexical correspondant qui ignore les caractères blancs.

```
/*
 * g2.c
 * Analyseur LL(2) récursif descendant pour la grammaire G2.
 * Exemple de compilation : `make g2`
 * Exemple d'utilisation : `echo "aab" | ./g2`
 */

#include <stdio.h>
#include <ctype.h>

/* compteurs pour les nombres de 'a' et de 'b' lus */
int nba, nbb;

/* valeur de retour :
   '0' sans erreur,
   '1' en cas d'erreur lexicale,
   '2' en cas d'erreur syntaxique. */
```

```
int retourerr;

/* lexemes courants */
int lexeme[2];

/* analyse les lexèmes :
   met 'lexeme[1]' dans 'lexeme[0]',
   et lit un nouveau lexème dans 'lexeme[1]'. */
static void
lexical (void)
{
  int l;

  if (lexeme[1] == EOF)
    lexeme[0] = EOF;
  else
    {
      do
        {
          l = getchar ();
          while (l != EOF && l != 'a' && l != 'b' && !isspace (l))
            {
              fprintf (stderr, "Lexème inconnu : %c\n", l);
              retourerr = 1;
              l = getchar ();
            }
        }
      while (l != EOF && isspace (l));

      lexeme[0] = lexeme[1];
      lexeme[1] = l;
    }
}

/* affiche une erreur syntaxique */
static void
erreur (char *attendus, int position)
{
  fprintf (stderr, "Erreur de syntaxe : on attendait %s, mais on a vu %c\n",
           attendus, lexeme[position]);
  retourerr = 2;
}

/* fonctions d'analyse syntaxique récursive */
static void S(void);
static void Aab(void);
static void Abb(void);

/* analyse S */
static void
S (void)
{
  switch (lexeme[0])
    {
    case 'a': /* S -> aAab */
      nba++;
      lexical ();
      Aab ();
```

```
      if (lexeme[0] == 'a')
        {
          nba++;
          lexical();
        }
      else
        erreur ("a", 0);
      if (lexeme[0] == 'b')
        {
          nbb++;
          lexical();
        }
      else
        erreur ("b", 0);
      break;
    case 'b': /* S -> bAbb */
      nbb++;
      lexical ();
      Abb ();
      if (lexeme[0] == 'b')
        {
          nbb++;
          lexical();
        }
      else
        erreur ("b", 0);
      if (lexeme[0] == 'b')
        {
          nbb++;
          lexical();
        }
      else
        erreur ("b", 0);
      break;
    default:
      erreur ("a ou b", 0);
    }
}

/* analyse A avec le contexte 'ab' */
static void
Aab (void)
{
  /* A -> a */
  if (lexeme[0] == 'a' && lexeme[1] == 'a')
    {
      nba++;
      lexical ();
    }
  /* A -> vide */
  else if (lexeme[0] != 'a')
    erreur ("a", 0);
  else if (lexeme[1] != 'b')
    erreur ("b", 1);
}

/* analyse A avec le contexte 'bb' */
static void
```

```
Abb (void)
{
  switch (lexeme[0])
    {
    case 'a': /* A -> a */
      nba++;
      lexical ();
      /* ici on pourrait vérifier que lexeme[1] est bien 'b',
         mais ce sera fait dans 'S ()'. */
      break;
    case 'b': /* A -> vide */
      break;
    default:
      erreur ("a ou b", 0);
    }
}

/* main */
int main (int argc, char *argv[]) {
  /* initialisations */
  nba = 0;
  nbb = 0;
  retourerr = 0;

  /* on remplit le tampon d'entrée */
  lexical (); lexical ();

  /* on vérifie que 'S' est bien reconnu */
  S ();
  if (lexeme[0] != EOF)
    erreur ("EOF", 0);

  if (retourerr == 0)
    printf ("Chaîne reconnue avec %d 'a' et %d 'b'.\n", nba, nbb);
  else
    printf ("Chaîne non reconnue avec %d 'a' et %d 'b'.\n", nba, nbb);
  return retourerr;
}
```

.....