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

Examen de juin 2005

Durée : 3 heures
Tous documents autorisés

Note : Les étudiants des parcours IM et IMT ne disposent que de deux heures pour cet examen, mais n'ont à traiter que les questions des parties 1, 2, 3 et 6, c'est-à-dire les questions a, b, c, d, e, f et l.

Grammaire G_0

Axiome = E
$N_0 = \{E\}$
$T_0 = \{n, -, /\}$
$P_0 = \{\ E \to n - E\ \}$
$E \to n/E$
$E \to n$

1 Grammaire linéaire à droite

Comme vous vous en souvenez, une grammaire linéaire à droite a toutes ses productions de la forme $A \to xB$ ou $A \to x$, où A et B sont dans N et x dans T^*. La grammaire G_0 est donc linéaire à droite.

Toute grammaire linéaire à droite (ou à gauche) engendre un langage rationnel.

a. Donnez une expression régulière décrivant $\mathcal{L}(G_0)$, le langage reconnu par G_0.

$$\mathcal{L}(G_0) = n((-|/)n)^*. \tag{1}$$

.....

b. Donnez, sous la forme de son diagramme de transition, un automate d'états finis déterministe reconnaissant $\mathcal{L}(G_0)$.

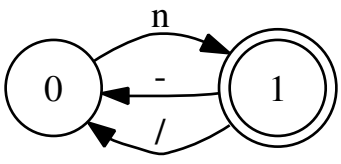

FIG. 1 – Automate d'états finis reconnaissant $\mathcal{L}(G_0)$.

.....

c. Donnez l'arbre de dérivation dans G_0 de la chaîne $n - n/n \vdash$. Que peut-on dire des contextes droits à chaque étape ?

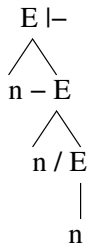

FIG. 2 – Arbre de dérivation pour la chaîne $n - n/n \vdash$.

Les contextes droits sont toujours réduits à $\vdash$.

. . . . . .

d. Montrez que, pour toute grammaire $G =< T, N, P, S >$ linéaire à droite et réduite, et pour tout non-terminal A de N, $SUIV_1(A) = \{\vdash\}$. Il est conseillé de procéder par induction sur le nombre n d'étapes d'une dérivation $S \Rightarrow_d^n \delta Ax$.

Soit la dérivation $S \Rightarrow_d^n \delta Ax$; montrons par induction sur le nombre n d'étapes de dérivation que $x = \nu$.

1. pour $n = 0$, on a forcément $A = S$ et $\delta = x = \nu$;
2. soit maintenant la dérivation

$$S \Rightarrow_d^{n-1} \beta Bz \Rightarrow_d \beta\alpha Ayz, \tag{2}$$

avec $B \to \alpha Ay$ dans P. Par hypothèse d'induction sur la dérivation $S \Rightarrow_d^{n-1} \beta Bz$,

$$z = \nu\ ; \tag{3}$$

et comme G est linéaire à droite,

$$y = \nu. \tag{4}$$

Dès lors, dans la dérivation $S \Rightarrow_d^n \beta\alpha Ayz$,

$$yz = \nu. \tag{5}$$

En utilisant ce résultat, il est clair que le seul contexte droit possible est la fin de fichier. □

. . . . . .

2 Analyse lexicale

Le lexème n de G_0 représente un entier écrit comme :
– soit le chiffre `0` pour l'entier zéro,
– soit un chiffre différent de zéro puis autant de chiffres que nécessaire pour les entiers en base dix.

e. Écrivez les règles flex permettant l'analyse lexicale pour G_0. La valeur retournée pour chaque lexème est un **`int`** nommé `N`, '`-`' ou '`/`' ; dans le cas où l'on a reconnu un entier, il convient aussi de mettre sa valeur dans `yylval`. Les blancs sont simplement ignorés. Vous disposez aussi d'une procédure **`void`** `yyerror (`**`char`** `*msg)` affichant un message d'erreur.

```
%%
0|[1-9][0-9]*  yylval = atoi (yytext); return N;
[\-/]          return *yytext;
[ \t\n]        ;
.              yyerror ("Lexème inconnu.\n");
%%
```

.....

3 Analyse LL

f. Écrivez en C un analyseur récursif descendant LL(1) qui reconnaît $\mathcal{L}(G_0)$. Vous disposez pour cela

- d'une fonction **int** `yylex (void)` renvoyant le prochain lexème lu,
- des lexèmes de type **int** nommés `N`, '−', '/' et `END`,
- du premier lexème lu dans une variable **int** `yychar`,
- d'une procédure **void** `yyerror (char *msg)` affichant un message d'erreur.

```
void
E (void)
{
  if (yychar == N)  /* E -> n Ep */
    {
      yychar = yylex ();
      Ep ();
    }
  else
    yyerror ("On attendait N.\n");
}

void
Ep (void)
{
  switch (yychar)
    {
    case '-': /* Ep -> '-' E */
    case '/': /* Ep -> '/' E */
      yychar = yylex ();
      E ();
      break;
    case END: /* Ep -> \nu */
      break;
    default:
      yyerror ("On attendait '-', '/' ou END.\n");
    }
}
```

.....

4 Analyse LR

g. Construisez l'automate LR(0) reconnaissant G_0. Ajoutez les contextes LALR(1) sur cet automate.

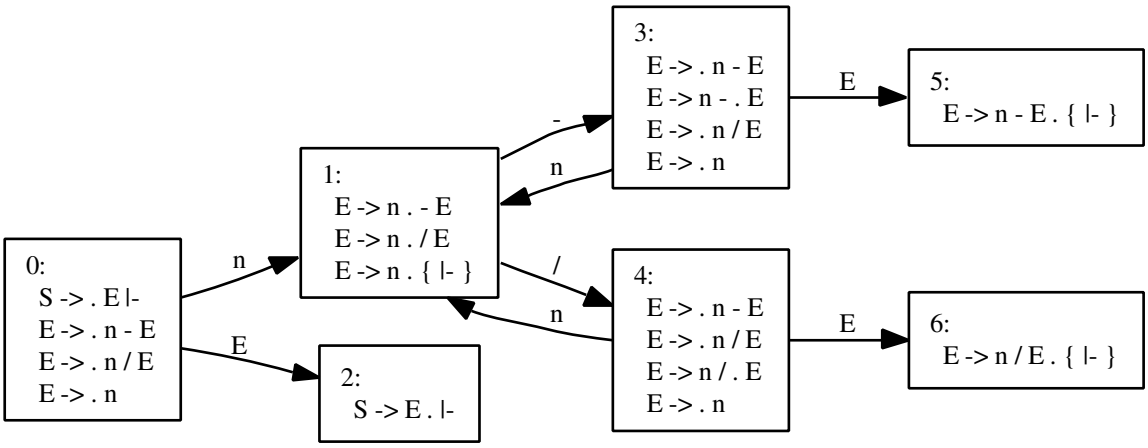

FIG. 3 – Automate LALR(1) reconnaissant G_0.

Comme G_0 est linéaire à droite, tous les contextes LR(1) valent $\{\vdash\}$.

. . . . . .

h. La grammaire G_0 est-elle

1. LR(0) ?
2. LALR(1) ?
3. SLR(1) ?
4. LR(1) ?

Justifiez à chaque fois votre réponse.

---

1. L'automate LR(0) de la figure 3 n'est pas déterministe : dans l'état 1, l'analyseur a le choix entre décalage et réduction de n à E ; G_0 n'est pas LR(0).
2. L'automate LALR(1) de la figure 3 est déterministe ; G_0 est LALR(1).
3. Comme G_0 est linéaire à droite,
$$\text{SUIV}_1(E) = \{\vdash\} \; ; \tag{6}$$
on retrouve les contextes droits de l'automate LALR(1) et G_0 est donc SLR(1).
4. Comme G_0 est SLR(1), elle est aussi LR(1).

. . . . . .

i. Écrivez le nécessaire pour faire générer un analyseur LALR(1) à Bison pour $\mathcal{L}(G_0)$. Vous prendrez soin d'imposer la priorité de la division sur la soustraction.

Deux solutions possibles :

1. une transformation de la grammaire

```
%token N
%%
e:
    e '-' t
  | t
  ;
```

```
t:
    t '/' N
  | N
  ;
%%
```

2. l'utilisation des directives de priorité des opérations de bison

```
%token N
%left '-'
%left '/'
%%
e:
    e '-' e
  | e '/' e
  | N
  ;
%%
```

Attention ! Les règles suivantes n'assurent pas la priorité de la division sur la soustraction :

```
%token N
%left '-'
%left '/'
%%
e:
    N '-' e
  | N '/' e
  | N
  ;
%%
```

En effet, comme G_0 est LALR(1), bison n'a jamais besoin d'utiliser les priorités pour résoudre des conflits.

. . . . . .

5 Analyse DR

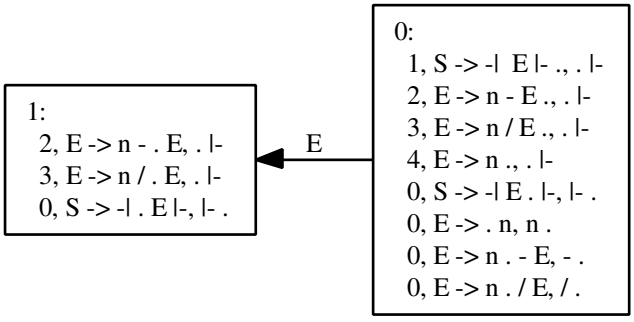

FIG. 4 – Analyseur DR(1) pour G_0.

j. Remplissez la table 1 d'analyse DR(1) correspondant à l'analyseur de la figure 4.

	$\vdash$	n	$-$	$/$	E	$\dashv$	$\vdash$	n	$-$	$/$
0		0	0	0	1		+1	+4		
1						0			+2	+3

TAB. 1 – Table d'analyse DR(1) pour G_0.

k. Montrez la trace de l'analyse de la phrase $n - n/n$. Soulignez à chaque étape la portion de la fenêtre et/ou de la pile utilisée pour décider de l'action.

pile	entrée	actions
$\dashv$	$\underline{n} - n/n \vdash$	décalage
$\dashv n$	$\underline{-}n/n \vdash$	décalage
$\dashv n-$	$\underline{n}/n \vdash$	décalage
$\dashv n - n$	$\underline{/}n \vdash$	décalage
$\dashv n - n/$	$\underline{n} \vdash$	décalage
$\dashv n - n/\underline{n}$	$\vdash$	réduction de n à E
$\dashv n - n/\underline{E}$	$\vdash$	réduction de n/E à E
$\dashv n\underline{-E}$	$\vdash$	réduction de $n - E$ à E
$\underline{\dashv E}$	$\vdash$	décalage
$\dashv E\underline{\vdash}$		accepte

6 Analyse sémantique

Le langage $\mathcal{L}(G_0)$ décrit des opérations arithmétiques :
- le lexème n synthétise une valeur entière,
- l'opérateur de division produit un résultat réel, quel que soit le type de ses opérandes,
- l'opérateur de soustraction produit un résultat entier si ses deux opérandes sont entiers, et sinon réel.

Les actions sémantiques doivent donc gérer les conversions nécessaires, et les attributs doivent permettre de faire ce travail convenablement.

l. Ajoutez à la grammaire G_0 les attributs et actions sémantiques nécessaires pour calculer la valeur d'une expression engendrée par cette grammaire.