École Polytechnique de Montréal
Département de génie informatique

INF4303 – Éléments de langage et compilateurs

Intra hiver 2003

Solutionnaire

Question 1 – Grammaire (4 points)

Soit la grammire:

$$\begin{array}{rcl} \mathit{assign} & \rightarrow & \mathit{lhs} \ = \ \mathit{rhs} \ ; \\ \mathit{lhs} & \rightarrow & \textbf{var} \mid * \ \textbf{var} \\ \mathit{rhs} & \rightarrow & \mathit{lhs} \mid * \ \mathit{rhs} \mid \textbf{num} \end{array}$$

(a) Quels sont les symboles terminaux de cette grammaire?

*Les jetons sont **var**, **num**, '*', '=' et ';'*

(b) Quels sont les non-terminaux?

Les non-terminaux sont assign, lhs et rhs

(c) Quel est le symbole de départ (symbole START)?

Le symbole de départ est assign

(d) Combien y a-t-il de règles de production?

Il y a 6 règles de production

(e) Montrez que la séquence de jetons suivante fait partie du langage décrit par la grammaire:

$$* \ \textbf{var} \ = \ \textbf{num} \ ;$$

La dérivation à gauche suivante le prouve:

$$\begin{array}{rcl} \mathit{assign} & \Rightarrow & \mathit{assign} \\ & \Rightarrow & \mathit{lhs} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ \textbf{num} \ ; \end{array}$$

(f) Montrez que la grammaire est ambiguë pour la chaîne suivante:

$$* \ \textbf{var} \ = \ * \ * \ \textbf{var} \ ;$$

On peut le démontrer en faisant deux dérivations à gauche (ou à droite) ou en dessinant deux arbres de parsage. Voici deux dérivations à gauche:

$$\begin{array}{rcl} \mathit{assign} & \Rightarrow & \mathit{assign} \\ & \Rightarrow & \mathit{lhs} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ \mathit{lhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ * \ \textbf{var} \ ; \end{array} \qquad \begin{array}{rcl} \mathit{assign} & \Rightarrow & \mathit{assign} \\ & \Rightarrow & \mathit{lhs} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ * \ \mathit{rhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ * \ \mathit{lhs} \ ; \\ & \Rightarrow & * \ \textbf{var} \ = \ * \ * \ \textbf{var} \ ; \end{array}$$

Question 2 – Construction d'un parseur descendant (6 points)

Voici une grammaire décrivant une partie du langage AWK:

$$\begin{array}{rcl} \mathit{prg} & \rightarrow & \mathit{stmt}\ \mathit{prg} \mid \mathit{stmt} \\ \mathit{stmt} & \rightarrow & \mathit{cond}\ \{\ \mathit{action}\ \} \mid \{\ \mathit{action}\ \} \\ \mathit{cond} & \rightarrow & \textbf{BEGIN} \mid \textbf{var relop num} \\ \mathit{action} & \rightarrow & \textbf{print string} \end{array}$$

(a) Écrivez le code d'un parseur descendant par appels de fonctions ("predictive parser") implantant cette grammaire. *Si nécessaire*, modifiez la grammaire. Vous pouvez coder dans le langage de votre choix.

Il faut d'abord enlever le préfixe commun pour les alternatives:

$$\mathit{prg} \rightarrow \mathit{stmt}\ \mathit{prg} \mid \mathit{stmt}$$

Par factorisation, j'obtiens ces trois nouvelles règles:

$$\begin{array}{rcl} \mathit{prg} & \rightarrow & \mathit{stmt}\ \mathit{prg}' \\ \mathit{prg}' & \rightarrow & \mathit{prg} \mid \epsilon \end{array}$$

Il n'y a pas d'autres modifications à apporter.

Je peux calculer les FIRST:

$$\begin{array}{rcl} \text{FIRST(action)} & = & \{\ \textbf{print}\ \} \\ \text{FIRST(cond)} & = & \{\ \textbf{BEGIN}, \textbf{var}\ \} \\ \text{FIRST(stmt)} & = & \{\ \textbf{BEGIN}, \textbf{var}, \text{'\{'}\ \} \\ \text{FIRST(prg)} & = & \{\ \textbf{BEGIN}, \textbf{var}, \text{'\{'}\ \} \end{array}$$

Et voici le code du parseur descendant:

```
 1 void prg()
 2 {
 3    stmt(); prg_prime();
 4 }
 5
 6 void prg_prime()
 7 {
 8    if (lookahead in {BEGIN, var, '{'}) {
 9       prg();
10    }
11    else {
12    }
13 }
14
15 void stmt()
16 {
17    if (lookahead in {BEGIN, var}) {
18       cond(); match('{'); action(); match('}');
19    }
20    else if (lookahead == '{') {
21       match('{'); action(); match('}');
22    }
23    else {
24       error();
25    }
26 }
27
28 void cond()
29 {
30    if (lookahead == BEGIN) {
31       match(BEGIN);
32    }
33    else if (lookahead == var) {
34       match(var); match(relop); match(num);
35    }
36    else {
37       error();
38    }
39 }
40
41 void action()
42 {
43    match(print); match(string);
44 }
```

(b) Dans quelle fonction le parseur débute-t-il?

Le parseur débute dans la fonction prg()

Question 3 – Exécution d'un parseur ascendant (4 points)

La grammaire de la question précédente est reprise à cette question. Elle a subi quelques transformations: elle a été augmentée (règle de production 0), les noms des symboles ont été abrégés et les règles ont été numérotées.

$$
\begin{array}{lrcl}
(0) & P' & \rightarrow & P \\
(1) & P & \rightarrow & S\ P \\
(2) & P & \rightarrow & S \\
(3) & S & \rightarrow & C\ \{\ A\ \} \\
(4) & S & \rightarrow & \{\ A\ \} \\
(5) & C & \rightarrow & \mathbf{B} \\
(6) & C & \rightarrow & \mathbf{v\ r\ n} \\
(7) & A & \rightarrow & \mathbf{p\ s}
\end{array}
$$

Les tables action et goto SLR construites pour cette grammaire sont présentées dans le tableau suivant.

Etat	action									goto			
	{	}	**B**	**v**	**r**	**n**	**p**	**s**	$	P	S	C	A
0	s4		s5	s6						1	2	3	
1									acc				
2	s4								r2	7	2	3	
3	s8												
4							s10						9
5	r5												
6					s11								
7									r1				
8							s10						12
9		s13											
10								s14					
11						s15							
12		s16											
13	r4		r4	r4					r4				
14		r7											
15	r6												
16	r3		r3	r3					r3				

La signification des actions est la même que dans le Dragon:

(1) sX signifie de décaler et d'empiler l'état X;

(2) rY signifie de réduire par la règle de production Y;

(3) acc signifie que la chaîne d'entrée est valide; le parseur arrête;

(4) une case vide signifie une erreur; le parseur arrête.

Simulez l'exécution du parseur LR utilisant ces tables et cette grammaire augmentée sur la séquence de jetons suivante:

B { **p** **s** } { **p** **s** $

Poursuivez la simulation jusqu'à ce que le parseur accepte la séquence de jetons ou annonce une erreur. Remplissez le tableau de cette page. Pour chaque étape, mettez un "x" pour indiquer sur quel jeton de la séquence d'entrée le *lookahead* pointe, indiquez le contenu de la pile (états et symboles parsés) et retranscrivez l'action du parseur (ex.: s4, r3, acc, err).

#	Entrée									Pile									Action
	B	{	**p**	**s**	}	{	**p**	**s**	$										
0	**x**									0									s5
1		**x**								0	**B**	5							r5
2		**x**								0	C	3							s8
3			**x**							0	C	3	{	8					s10
4				**x**						0	C	3	{	8	**p**	10			s14
5					**x**					0	C	3	{	8	**p**	10	**s**	14	r7
6					**x**					0	C	3	{	8	A	12			s16
7						**x**				0	C	3	{	8	A	12	}	16	r3
8						**x**				0	S	2							s4
9							**x**			0	S	2	{	4					s10
10								**x**		0	S	2	{	4	**p**	10			s14
11									**x**	0	S	2	{	4	**p**	10	**s**	14	err
12																			
13																			
14																			
15																			
16																			
17																			
18																			

Question 4 – Construction des ensembles d'items (4 points)

La grammaire de la question précédente est reprise à cette question. Quelques règles de production ont été enlevées.

$$
\begin{array}{lrcl}
(0) & P' & \rightarrow & P \\
(1) & P & \rightarrow & S\ P \\
(2) & P & \rightarrow & S \\
(3) & S & \rightarrow & \{\ A\ \} \\
(4) & A & \rightarrow & \mathbf{p}\ \mathbf{s}
\end{array}
$$

Construisez les ensembles d'items. Notez que la grammaire a déjà été augmentée (règle de production 0).

Voici les 9 ensembles d'items:

I_0 :
$P' \rightarrow \cdot P$
$P \rightarrow \cdot S\ P$
$P \rightarrow \cdot S$
$S \rightarrow \cdot\{\ A\ \}$

I_1 : $P' \rightarrow P\cdot$

I_2 :
$P \rightarrow S \cdot P$
$P \rightarrow S\cdot$
$P \rightarrow \cdot S\ P$
$P \rightarrow \cdot S$
$S \rightarrow \cdot\{\ A\ \}$

I_3 :
$S \rightarrow \{\ \cdot A\ \}$
$A \rightarrow \cdot \mathbf{p}\ \mathbf{s}$

I_4 : $P \rightarrow S\ P\cdot$

I_5 : $P' \rightarrow \{\ A\cdot\ \}$

I_6 : $P \rightarrow \mathbf{p} \cdot \mathbf{s}$

I_7 : $S \rightarrow \{\ A\ \}\cdot$

I_8 : $S \rightarrow \mathbf{p}\ \mathbf{s}\cdot$

Question 5 – Traduction dirigée par la syntaxe (2 points)

Pour la grammaire de la question précédente, écrivez une définition dirigée par la syntaxe (SDD) qui calcule le nombre d'instructions dans un programme AWK (représentées par le non-terminal S) et affiche ce nombre une seule fois. L'affichage se fait par l'appel à la fonction *print*, qui accepte un paramètre, le nombre à afficher. Aucun autre effet de bord (appel de fonctions ou référence à des variables globales) n'est permis. Utilisez le tableau suivant.

Règles de production	Actions sémantiques
(0) $P' \rightarrow P$	$print(P.s)$
(1) $P \rightarrow S\ P_1$	$P.s = 1 + P_1.s$
(2) $P \rightarrow S$	$P.s = 1$
(3) $S \rightarrow \{\ A\ \}$	
(4) $A \rightarrow$ **p s**	