Licence d'Informatique 2003-2004 **Langages et Automates**

TD N° 8

ANALYSE SYNTAXIQUE DESCENDANTE (SUITE)

Exercice 1 : Elimination de la récursivité gauche

Eliminer la récursivité gauche dans la grammaire G1 suivante (expressions arithmétiques associant à gauche) :

E → E + T | T
T → T * F | F
F → (E) | id

Exercice 2

Encore une grammaire (G2) d'expressions arithmétiques, additives cette fois mais avec un "-" unaire et un "-" binaire, et associant à droite.

E → F + E | F - E | F
F → - F | (E) | id

N.B. G2 est non ambiguë.

- G2 a-t-ellr une chance d'être LL(1) ? Sinon, donner une grammaire G'2 équivalente qui le soit (peut-être).

- Définir les fonctions PREMIER et SUIVANT et la table d'analyse pour G'2

- Simuler l'analyseur déterministe pour : - id - id et - id - + id

Exercice 3

Soit la grammaire G3 (S-expressions Lisp) :

S → (L) | () | at
L → S | L S

- G3 est-elle LL(1). Sinon la transformer pour qu'elle le soit.

- Calculer la table d'analyse

Exercice 4

Soit G4 (expressions conditionnelles) :

I → if E alors I sinon I | si E alors I | a
E → b

(a pour "autre", b pour "expressions booléennes", non analysées ici)

Peut-on mettre G4 sous forme LL(1). Peut-on néanmoins "déterminiser l'analyse" - par un dispositif "ad hoc" ? interprétation ?

Exercice 5

Formulez un ensemble de conditions nécessaires et suffisantes pour qu'une grammaire soit LL(1). Indication : considérez les couples de règles A —> w1, A —> w2 et trouver une condition su PREMIER et SUIVANT.

Licence d'Informatique 2003-2004 **Langages et Automates**

TD N° 8

ANALYSE SYNTAXIQUE DESCENDANTE (SUITE) - CORRIGE

Elimination de la Récursivité gauche

Exercice 1

Il s'agit ici d'éliminer la récursivité gauche directe. On applique les règles du cours.

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T'| ε
F → (E) | id

qui associe toujours à gauche, mais sans récursivité gauche.

Exercice 2

- G2 n'est pas LL(1) : PREM(F+E) = PREM(F-E) = PREM(F) = { id, -, (}. Donc on aura les trois règles pour E dans M(E,x), pour x égal à un de ces 3 terminaux.

- Solution: factorisation -> G'2 :

E → F E'
E' → + E | - E | ε
F → - F | (E) | id
(où l'on fait apparaître un ε production)

- PREMIER :
  - Eff(G'2) = {E'} (variables effaçables)
  - PREM(E) = PREM(F)
  - PREM(E') = {+,-}
  - PREM(F) = {-, (, id}

- SUIVANT
  - au départ : SUIV(E) = { $ }
  - Règle 2 : SUIV(F) contient PREM(E')= {+,-} et SUIV(E) contient), donc devient {), $})
  - Règle 3 : on ajoute SUIV(E) à SUIV(E'), donc SUIV(E') devient {), $ }. Et on ajoute SUIV(E) à SUIV(F) puisque E' est effaçable. SUIV(F) devient {+,-,), $}

Au total : SUIV(E) = SUIV(E') = {), $ }, et SUIV(F) = {+,-,), $}

- TABLE D'ANALYSE

	+	-	(	)	id	$
E		E —> F E'	E —> F E'		E —> F E'	
E'	E' —> + E	E' —> - E		E' —> ε		E' —> ε
F		F —> - F	F —> (E)		F —> id	

- Analyse de : - id - id

Entrée	Pile	Opération/règle
- id - id $	E$	E —> F E'
- id - id $	FE'$	F —> - F
id - id $	-FE'$	dépiler -
id - id $	FE'$	F —> id
id - id $	idE'$	dépiler id
- id $	E'$	E' —> - E
- id $	-E $	dépiler -
id $	E $	E —> F E'
id $	FE'$	F —> id
id $	idE'$	dépiler id
$	E'$	E' —> ε
$	$	SUCCES

- Analyse de : - id - + id
Jusqu'à la dernière ligne, c'est la même chose.

Entrée	Pile	Opération/règle
- id - + id $	E$	E —> F E'
- id - + id $	FE'$	F —> - F
id - + id $	-FE'$	dépiler -
id - + id $	FE'$	F —> id
id - + id $	idE'$	dépiler id
- + id $	E'$	E' —> - E
- + id $	-E $	dépiler -
+ id $	E $	ERREUR

Remarque : on peut imaginer un message d'erreur simple du type :
attendu : - ou (ou id trouvé : +
Ou encore :
attendu : phrase de type E

Exercice 3

Soit G3 (S-expressions Lisp) :
S —> (L) | () | at
L —> S | L S

Deux problèmes : 1) G3 est récursive à gauche (récursion directe sur L) et 2) PREM((L)) et PREM(()) contiennent tous deux (.

- Elimination de la récursivité gauche
S —> (L) | () | at
L —> S L'
L' —> S L' | ε

- Déterminisation,: par factorisation (G'3)
S —> (S' | at
S' —> L) |)
L —> S L'
L' —> S L' | ε

Eff(G'3) = {L'}

PREM(S) = {(,at}
PREM(S') = PREM(L) U {)} = {(,),at}
PREM(L) = PREM(L') = PREM(S) = {(,at}

SUIV(S) = {$} axiome
U PREM(L') U SUIV(L) à cause de L—> S L'
U PREM(L') U SUIV(L') à cause de L' —> S L'
= {$, (, at} U SUIV(L) U SUIV(L')
SUIV(S') = SUIV(S)
SUIV(L) = {)}
SUIV(L') = SUIV(L)

	PREMIER	SUIVANT
S	(at	( ) at
S'	( ) at	( ) at
L	(at	)
L'	(at	)

- Table d'analyse

	(	)	at	$
S	S —> (S'		S —> at	
S'	S' —> L)	S' —>)	S' —> L)	
L	L —> S L'		L —> S L'	
L'	L' —> S L'	L' —> ε	L' —> S L'	

Exercice 4

G4 :
I —> si E alors I sinon I | si E alors I | a
E —> b

On peut essayer de factoriser "si E alors I":
G'4 :
I —> si E alors I I' | a
I' —> sinon I | e
E —> b

Eff(G'4) = {I'}

PREM(I) = {si,a}
PREM(I') = {sinon}
PREM(b) ={b}

SUIV(I) = {$} U PREM(I') U SUIV(I') = {$, sinon}
SUIV(I') = SUIV(I)
SUIV(E) = {alors}

	a	b	si	alors	sinon	$
I	I —> a		I —> si E alors I I'			
I'					I' —> sinon I I' —> ε	I' —> ε
E		E —> b				

La grammaire n'est donc pas LL(1).

Solution : décider que lon préférera systématiquement la règle I' —> sinon I, ce qui correspond à associer le sinon avec le alors le plus proche, comme déjà vu par une autre méthode dans le TD n° 4.

Exercice 5

(a) Pour tout couple de productions A—>w1 et A—>w2, PREM(w1) et PREM(w2) sont disjoints.

(b) Si A est effaçable pour toute production A—>w, PREM(w) et SUIV(A) sont disjoints.