Université Grenoble Alpes **UFR IM²AG**
DU ISN-C2.3 **Année 2019-20**

Langages et Compilation : analyse syntaxique

Langage d'expressions arithmétiques

1 Le langage

Le lexique est composé d'entiers naturels, des deux opérateurs `+` et `*`, et de parenthèses. Les opérateurs sont associatifs à gauche, l'opérateur `*` est plus prioritaire que l'opérateur `+`. On dénote les entiers par le symbole terminal `n`.

Le langage peut être décrit par la grammaire suivante :

V_t = { n, (,), *, +, finentrée}
V_N = { Z, E, T, F }

Z	: :=	E finentrée
E	: :=	E + T | T
T	: :=	T * F | F
F	: :=	n | (E)

2 Analyse syntaxique descendante

La grammaire proposée étant récursive à gauche, elle ne permet pas une analyse syntaxique de type LL (descendante).

Après élimination de la récursivité à gauche, on obtient la grammaire suivante :

V_t = { n, (,), *, +, finentrée }
V_N = { Z, E, T, F, ST, SF }

Z	: :=	E finentrée
E	: :=	T ST
ST	: :=	ε | + T ST
T	: :=	F SF
SF	: :=	ε | * F SF
F	: :=	n | (E)

Le calcul des directeurs pour chaque règle donne :

directeur (Z : := E finentrée)	=	premier (E) = { n, (}
directeur (E : := T ST)	=	premier (T) = premier (F) = { n, (}
directeur (ST : := ε)	=	suivant (ST) = suivant (E) = {), finentrée }
directeur (ST : := + T ST)	=	{ + }
directeur (T : := F SF)	=	premier (F) = { n, (}
directeur (SF : := ε)	=	suivant (SF) = suivant (T) = premier (ST) = suivant (ST) $\cup$ { + } = {), finentrée, + }
directeur (SF : := * F SF)	=	{ * }

directeur (F : := n)	=	{ n }
directeur (F : := (E))	=	{ (}
directeur (ST : := ε) $\cap$ directeur (ST : := + T ST)	=	$\emptyset$
directeur (SF : := ε) $\cap$ directeur (SF : := * F SF)	=	$\emptyset$
directeur (F : := n) $\cap$ directeur (F : := (E))	=	$\emptyset$

Les intersections des ensembles directeurs associés à des règles formant des alternatives pour un même symbole non-terminal étant vides, la grammaire est LL(1).

3 Construction d'un interpréteur

On peut décrire des traitements en s'appuyant sur la structure de la grammaire. Par exemple on veut construire un programme qui calcule la valeur d'une expression arithmétique décrite par la grammaire précédente.

Pour cela, on associe à chaque fonction de reconnaissance une valeur entière qui est la valeur de la sous-expression reconnue. Ces fonctions de reconnaissance renvoient donc un résultat de type entier.

La difficulté reside dans le fait que, dans la grammaire transformée pour les besoins de l'analyse LL(1), les sous-expressions sont "eparpillées".

Les valeurs des sous-expressions calculées dans certains sous-arbres de l'arbre de dérivation doivent être "transmises" à d'autres sous-arbres. On ajoute donc des paramètres aux fonctions de reconnaissance qui peuvent recevoir en entrée les valeurs de sous-expressions calculées ailleurs dans l'arbre.

Considérons l'expression : 2 * 3 + 4

La figure ci-dessous montre l'arbre de dérivation de cette expression et les résultats calculés et transmis par chaque noeud de cet arbre.