

Notes de cours

GIF 340 : ÉLÉMENTS DE COMPILATION

Chapitre 4

Arbres syntaxiques abstraits

Ahmed KHOUMSI

Introduction

Analyseur lexical (scanner) fournit une séquence d'unités lexicales (terminaux, tokens) à l'analyseur syntaxique

Analyseur syntaxique (parser) :

1. Regroupe les unités lexicales en structures grammaticales
2. Vérifie si la syntaxe (grammaire) du programme est correcte
3. Génère un arbre syntaxique abstrait qui représente la structure hiérarchique du flot d'unités lexicales

Le point 1 est nécessaire pour effectuer les points 2 et 3.

Les points 2 et 3 constituent le résultat du parser

Nous allons présenter en détail les arbres syntaxiques abstraits

Arbres syntaxiques abstraits (AST = abstract syntax tree)

Un arbre syntaxique abstrait sera plus simplement appelé arbre abstrait et noté AST (Abstract Syntax Tree)

Un AST :

- contient des nœuds et des feuilles
- est une forme de représentation interne des caractéristiques essentielles d'un programme source

Feuille d'un AST correspond à un objet :

- sur lequel un opérateur est appliqué et
- qui n'est pas un résultat d'opération

Nœud d'un AST

- Nœud d'opérateur
 - correspond à un opérateur arithmétique (+ - * / & ...) ou implicite (indexation de tableau) appliqué à des opérandes qui peuvent être :
 - des feuilles d'AST
 - des nœuds d'AST (concrètement, des résultats d'opérations)
 - Représente le résultat de l'opération
- Nœud d'instruction correspond à une occurrence d'instruction

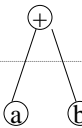
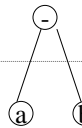
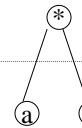





Exemples d'ASTs

Feuilles

Une feuille peut correspondre à

- un identificateur (unité lexicale *id*)
- une valeur entière (unité lexicale *nb*)
- une valeur booléenne (unité lexicale *bo*)
- une valeur de type chaîne de caractères (unité lexicale *ch*)



Nœuds d'opérateurs

Opération	$a + b$	$a - b$	$a * b$	a / b
Nœud d'opérateur				
Feuilles				

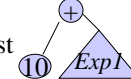

Exemples d'ASTs (suite)

On considère l'instruction d'affectation $p := 10 + v * t$;

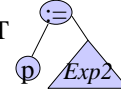
1. On considère l'expression $v * t$ que l'on désignera par *Exp1*

l'AST correspondant est  désigné aussi par 

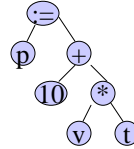
2. On considère l'expression $10 + \text{Exp1}$ que l'on désignera par *Exp2*

l'AST correspondant est  désigné aussi par 

3. On considère l'expression $p := \text{Exp2}$ représentée par l'AST

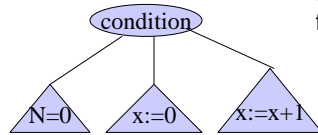


En faisant les remplacements adéquats, on obtient finalement l'AST

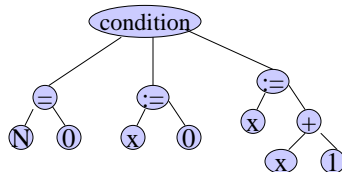


Exemples d'ASTs (suite)

On considère l'instruction de condition $\text{si } (N = 0) \text{ alors } x := 0;$
 $\text{sinon } x := x + 1;$
 fin si

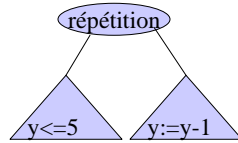


En remplaçant les trois nœuds par les ASTs qui les représentent, on obtient

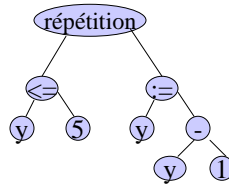


Exemples d'ASTs (suite)

On considère l'instruction de répétition `tantque (y <= 5) faire`
`y := y - 1;`
`fintq`



En remplaçant les deux nœuds par les ASTs qui les représentent, on obtient



Évaluation récursive d'AST d'expression arithmétique

On considère un AST représentant une expression arithmétique constitué de feuilles et de nœuds tels que :

- une feuille a une valeur
- un nœud est défini par un opérateur binaire et ses enfants gauche et droit

Voici une méthode récursive d'Évaluation d'AST :

- Évaluation d'une feuille = valeur de la feuille
- Évaluation d'un nœud = valG Op valD, où :
 Op est l'opérateur du nœud
 valG est l'Évaluation de l'enfant gauche de N
 valD est l'Évaluation l'enfant droit de N
- Évaluation d'un AST = Évaluation de sa racine.

Évaluation récursive d'AST d'expr. arithm. (exemple)

$$\text{Eval}(R) = \text{Eval}(N1) - \text{Eval}(N7)$$

$$\text{Eval}(N1) = \text{Eval}(F1) - \text{Eval}(F2) = 1 - 2$$

$$\text{Eval}(N7) = \text{Eval}(N4) / \text{Eval}(N6)$$

$$\text{Eval}(N4) = \text{Eval}(N3) * \text{Eval}(F6) = \text{Eval}(N3) * 6$$

$$\text{Eval}(N3) = \text{Eval}(F3) * \text{Eval}(N2) = 3 * \text{Eval}(N2)$$

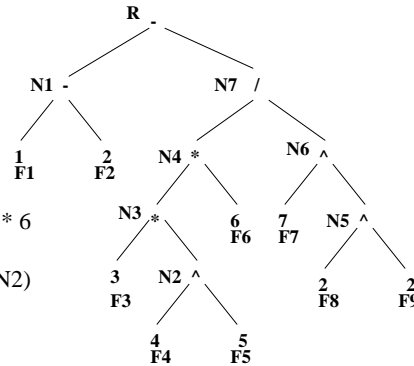
$$\text{Eval}(N2) = \text{Eval}(F4) ^ \text{Eval}(F5) = 4 ^ 5$$

$$\text{Eval}(N6) = \text{Eval}(F7) ^ \text{Eval}(N5) = 7 ^ \text{Eval}(N5)$$

$$\text{Eval}(N5) = \text{Eval}(F8) ^ \text{Eval}(F9) = 2 ^ 2.$$

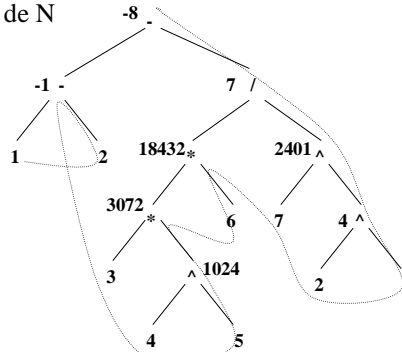
Résultat final : Eval(R) = -8

Résultat final : $((1 - 2) - ((3 * (4 ^ 5)) * 6) / (7 ^ (2 ^ 2)))$



Évaluation d'AST d'expr. arithm. par un parcours postfix

- parcourir les feuilles de gauche à droite
- remonter à un nœud à partir de son enfant le plus à droite (càd lorsque tous les enfants du nœud ont été parcourus)
- s'arrêter lorsqu'on arrive au nœud racine.
- Lorsqu'on arrive à une feuille, on l'évalue
- Lorsqu'on arrive à un nœud N, on l'évalue en appliquant l'opérateur de N aux évaluations des enfants de N



Lecture d'AST d'expression arithmétique

Il s'agit de traduire une structure de données d'AST sous une forme lisible (sur l'écran ou sur un fichier).

Une utilité est de pouvoir tester si un AST a été correctement construit.

On peut utiliser des procédures plus ou moins complexes de représentation d'AST.

On considère le cas simple d'AST constitué de feuilles et de nœuds tels que :

- une feuille est définie par une chaîne de caractères (nombre ou identificateur)
- un nœud est défini par un opérateur binaire et ses enfants gauche et droit

Voici une méthode réursive de Lecture d'AST :

- Lecture d'une feuille = impression de la chaîne de caractères de la feuille
- Lecture d'un nœud consiste à faire les opérations suivantes dans l'ordre :
 - Impression d'une parenthèse ouvrante
 - Lecture de l'enfant gauche du nœud
 - impression de l'opérateur du nœud
 - Lecture de l'enfant droit du nœud
 - Impression d'une parenthèse fermante
- Lecture d'un AST = Lecture de sa racine.

Lecture récursive d'AST d'expr. arithm. (exemple)

On considère l'AST de la page 9 :

```

Lect(R) = "(" + Lect(N1) + "-" + Lect(N7) + ")"
Lect(N1) = "(" + Lect(F1) + "-" + Lect(F2) + ")"
Lect(F1) = "1"
Lect(F2) = "2"
Lect(N7) = "(" + Lect(N4) + "/" + Lect(N6) + ")"
Eval(N4) = "(" + Lect(N3) + "*" + Lect(F6) + ")"
Lect(N3) = "(" + Lect(F3) + "*" + Lect(N2) + ")"
Lect(F3) = "3"
Lect(N2) = "(" + Lect(F4) + "^" + Lect(F5) + ")"
Lect(F4) = "4"
Lect(F5) = "5"
Lect(F6) = "6"
Eval(N6) = "(" + Lect(F7) + "^" + Lect(N5) + ")"
Lect(F7) = "7"
Eval(N5) = "(" + Lect(F8) + "+" + Eval(F9) + ")"
Lect(F8) = "2"
Lect(F9) = "2"
    
```

Résultat final : $((1 - 2) - ((3 * (4^5)) * 6) / (7^{(2^2)}))$

Feuille d'AST représentant un identificateur

Nous proposons

Ident
$\wedge Nom$
Type
$\wedge Decl$

Où $\wedge Nom$ pointe sur chaîne de caractère contenant nom de l'identificateur

Type déterminé à partir de la déclaration de l'identificateur, normalement mis à jour lors de l'analyse sémantique

$\wedge Decl$ pointe sur début de liste des déclarations

- locales, si l'identificateur est utilisé dans une procédure
- globales, si l'identificateur est utilisé dans le programme principal

Exemple : vitesse

S'il s'agit d'une variable ou constante déclarée comme étant entière

Ident
entier

→ vitesse

→ Début de liste où la variable ou constante est déclarée

Feuilles d'AST représentant un entier, une chaîne et un booléen

Entier Nous proposons

Nombre
Valeur

Exemple : nombre 3257 représenté par

Nombre
3257

Chaîne de caractères Nous proposons

Chaîne
$\wedge Ch$

Exemple : chaîne "bonjour" représentée par

Chaîne
→ bonjour

Booléen Nous proposons

Booleen
Valeur

Exemple : valeur vrai représentée par

Booleen
vrai

AST représentant une opération binaire

Nous proposons

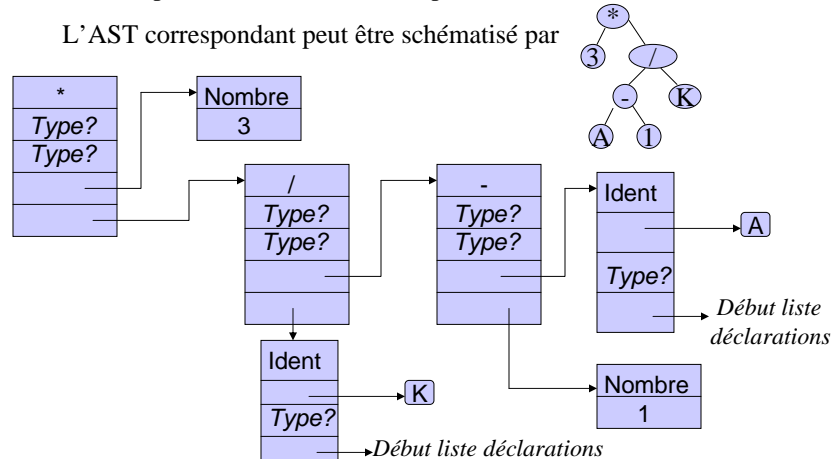
Opérateur
Type résultat
Type opérandes
^Opérande de gauche
^Opérande de droite

Opérateur spécifie l'opérateur
 Type résultat spécifie le type du résultat
 Type opérandes représente le type des deux opérandes, normalement mis à jour lors de l'analyse sémantique
 ^Opérande de gauche pointe sur AST représentant opérande de gauche
 ^Opérande de droite pointe sur AST représentant opérande de droite

AST représentant une opération binaire (suite)

À titre d'exemple, nous considérons l'expression $3 * ((A - 1) / K)$

L'AST correspondant peut être schématisé par



Type? normalement déterminés lors de l'analyse sémantique, correct si entier