Arbres abstraits

Arthur Garnier

1 Où est-on ?

2 But de l'analyse sémantique

Vérifier que le programme a un sens, décrit par un certain nombre de règles de sémantiques. Ces dernières étant données par le manuel de référence du langage.

2.1 Exemples de règles sémantiques :

- Variables déclarées avant utilisation
  - A chaque utilisation, trouver la déclaration associée
- Cohérence de types
- Double déclaration
- Nombre de paramètre formel = nombre de paramètre effectif
- Cohérence de type entre les paramètres formels et effectifs
- if(bool)

\

- L'analyse sémantique utilise l'AST (Abstract Syntax Tree) pour effectuer les controles sémantique.
- L'AST est construit **pendant** l'analyse syntaxique (via des fonctions sémantiques).
- L'AST est un résutat de l'analyse syntaxique
- L'AST = l'arbre syntaxique dans lequel on a supprimé tous les non terminaux de certains terminaux inutiles (;, (,), {, })

2.2 Passage de l'Arbre Syntaxique à l'Arbre Abstrait

L'AST est un arbre syntaxique dans lequel on supprime tout ce qui est inutile pour la **génération de code** → Les contrôles sémantiques.

L'AST (sous ANTLR) sera constuit par des "règles de ré-écritures" dans la grammaire.

3 Construire des AST sous ANTLR (Avec des règles de ré-écriture)

3.1 Supprimer des tokens (unités lexicales)

En général : (),;... sont inutiles pour la génération de code ⇒ donc inutiles dans l'AST

```
exp : '(' exp ')' -> exp
\| INT -> INT
;
```

Table des symboles (à savoir pour le partiel):

“x”	Type	Taille/Déplacement	Nature (var/param)
y			
z			
g	function	nb param	val retour

/! Par défaut ANTLR crée un AST “à plat” qui correspond à une liste chainée de token.

3.2 Réarrangement

```
Decl : 'var' ID ':' type -> type ID
;
```

var et : sont supprimés

3.3 Des racides d’AST

Pour avoir une structure 2D, il faut indiquer quels tokens doivent devenir des racines d’autres sous-arbres.

Syntaxe :

```
-> ^(root fils1+ fils2* fils3 ...)

statement : 'return' expr '*' -> ^('return' expr)
;
decl : 'var' ID ':' type -> ^('var' type ID)
;
```

3.4 Noeuds imaginaires

```
decl : type ID ';' -> ^(VAR type ID)
;
```

3.5 Collecte de tokens

```
list : ID (',' ID)* -> ID+
;
```

L’AST est une liste d’ID sans ‘,’ (liste chainée de noeuds ID)

```
decl : 'int' ID (',' ID)* ';' -> ^('int' ID+)
decl : 'int' ID(',' ID*) ';' -> ^('int' ID)+
```

Avoir plusieurs arbres de la forme ^(‘int’ ID)

Dans cet exemple :

int x,y;

^(‘int’ x)

^(‘int’ y)

3.6 Cas des labels

```
for_statement : 'for' '(' decl? ';' cond=expr? ';' iter=expr ')' list_inst
;
-> ^(FOR decl? ^(CONDITION $cond)? ^(ITER $iter) list_inst)
```