TD - TDL

Louis Thevenet

Table des matières

1. TD3

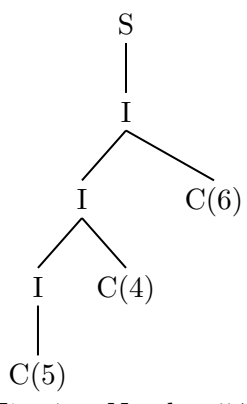


Fig. 1. – Nombre 546

Quel attribut faut-il associer aux symboles de cette grammaire ?

Attribut valeur pour le non terminal I: synthétisé des feuilles vers racine et de type $\mathbb{N}$

0
$\to 5$
$\to 5 \times 10 + 4$
$\to 54 \times 10 + 6$

La règle 3 est récursive à gauche, la grammaire n'est pas $\mathcal{LL}(k)$

Eliminons la récursivité à gauche :

$$\begin{cases} I \underset{a}{\to} cX \\ X \underset{b}{\to} cX \\ X \underset{c}{\to} \Omega \end{cases}$$

Fig. 2. – Remplace (2) et (3)

Réécrivons 546:

$$\begin{aligned} & S \\ & \underset{1}{\Rightarrow} I \\ & \underset{a}{\Rightarrow} c_5 X \\ & \underset{b}{\Rightarrow} c_5 c_4 X \\ & \underset{b}{\Rightarrow} c_5 c_4 c_6 X \\ & \Rightarrow c_5 c_4 c_6 \end{aligned}$$

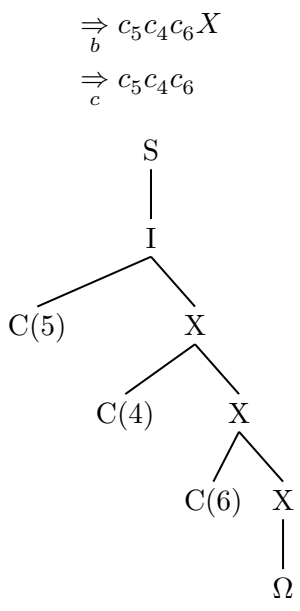


Fig. 3. – Nombre 546 sans récursivité à gauche

Attributs pour I et X ?

- val $\in \mathbb{N}$ synthétisé pour I et X
- exp $\in \mathbb{N}$ synthétisé pour X

2. TD4

La table des symboles permet de créer un lien entre les définitions et les utilisations des éléments nommés.

```
1  test {
2    int i = 1;
3    const int j = 2;
4    <int, int> p = <3, 4>;
5    int k = fst p;
6    if ( i < 5 ) {
7      int j = 5;
8      j = i * (snd p);
9      i = j + 1;
10     while ( k < 10 ) {
11       int p = 3;
12       k = k + i;
13     }
14   } else {
15     if ( i + j > 10 ) {
16       const boolean p = false;
17       print p;
18     }
19     print p;
20   }
21   print j;
22 }
```

Par exemple i est un élément nommé qu'on utilise par la suite.

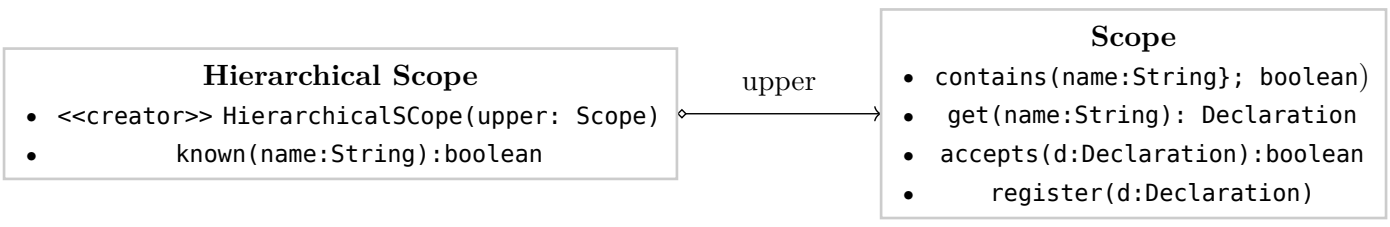
A chaque référence à cette variable i, on va vouloir faire référence à sa définition.

Le j ligne 7 est une redéfinition.

code	variables	actions
$\emptyset$		
`int i = 1;`	{i}	vérifier que la définition est autorisée, ajouter la définition i
`const int j = 2;`	{i, j}	vérifier que la définition est autorisée, ajouter la définition j
`<int, int> p = <3, 4>;`	{i, j, p}	vérifier que la définition est autorisée, ajouter la définition p
`int k = fst p;`	{i, j, k, p}	vérifier que p est définie; vérifier que la définition de k est autorisée, ajouter la définition p
`if ( i < 5 ) {`	{i, j, k, p}	Vérifier que i est bien définie
`int j = 5;`	{i, j, k, p}	La définition est interne au sous-bloc donc on a le droit de redéfinir, ajouter la définition de j
`int j = 5;`	{i, j, k, p}, {j}	La définition est interne au sous-bloc donc on a le droit de redéfinir, ajouter la définition de j

Tableau 1. – Lignes 1 à 5

On va créer de nouvelles tables de symboles à chaque fois qu'on entrera dans un nouveau bloc.



- $B \to \{L_I\}$
  1. L_I.tds $=$ new SymbolTable(B.tds)
     attribut hérité
  2. ```
     class Block {
       List<Instruction> instructions;
       boolean collect(Scope<Declaration> tds) {
         SymbolTable<Declaration> local = new SymbolTable(tds);

         boolean success = true;
         for (Instruction i : this.instructions) {
           success = success && i.collect(local);
           if (!success) {
             break;
           }
         }
         return success;
       }
     }
     ```
- $L_I \to \#1 I L_{I_1}$
  1. $\begin{cases} I.\text{tds} = L_I.\text{tds} \\ \underbrace{L_{I_1}.\text{tds}}_{\text{(attributs hérités)}} \end{cases} = L_I.\text{tds})$
     $\#1$
- $I \to \frown$ while $(E) B$
  1. $\begin{cases} E.\text{tds} = I.\text{tds} \\ \underbrace{(B.\text{tds})}_{\text{(attributs hérités)}} = I.\text{tds} \end{cases}$
  2. ```
     class Repetition implements Instruction {
       boolean collect(Scope<Declaration> tds){
         return condition.collect(tds)
         && body.collect(tds);
       }
     }
     ```
- $I \to T$ ident $= E$;
  ▸ ```
    class VariableDeclaration implements Instruction {
      boolean collect(Scope<Declaration> tds){
        if (I.tds.accepts(this)){
          I.tds.register(this);
          return true;
        } else {
          return false;
        }
      }
    }
    ```
- $E \to$ Ident
  1. Si `E.tds.knows(identificateur)` Alors lier utilisation à la déclaration Sinon signaler une erreur
  2. ```
     class VariableUse implements Instruction {
       boolean resolve(Scope<Declaration> tds) {
         if (tds.knows(this.name)) {
           this.declaration = tds.get(this.name);
           return true;
         } else {
           return false;
         }
       }
     }
     ```