SEMC

PRÉSENTATION SUCCINCTE

Pascal Bouchon
3ème année ENSIMAG
Grenoble, France

Juin 1988

1 Introuction

SEMC est un processeur de SYNTAX écrit par Bernard Lohro à l'INRIA Rocquencourt dans les années 1980. Ce processeur s'appelait alors TABC ; il a été renommé en SEMC au mois de mai 2023.

La sémantique effectuée par attributs synthétisés est toujours décrite dans un fichier `nom_du_langage.semc`. Dans ce fichier, le programmeur doit décrire les règles de grammaire selon la convention du module BNF. Après chaque règle de grammaire, le programmeur a la possibilité de décrire des actions synthétisées programmées en langage C. Ces actions sont exécutées pendant l'analyse syntaxique à chaque réduction.

2 Actions synthétisées

Les actions synthétisées suivant une règle de grammaire sont toujours exécutées lorsque la partie droite de la règle a été reconnue au moment de la réduction de cette règle.

Soit la grammaire suivante :

```
<axiome> = <liste> ;

<liste>  = <liste> , <identificateur> ;

<liste>  = <identificateur> ;

<identificateur> = "ok" ;
           printf("lu");
```

Après la reconnaissance du mot `"ok"`, au moment de réduire la règle

```
<identificateur> = "ok"
```

SEMC exécutera la séquence C correspondant à l'action associée à cette règle c'est-à-dire `printf("lu")`.

3 Attributs synthétisés

SEMC permet la gestion entièrement transparente pour le programmeur d'attributs synthétisés.

Ces attributs sont soumis à des conditions d'utilisation afin de permettre à SEMC de vérifier la cohérence de leurs emplois.

Ces attributs sont des identificateurs de variables declarés en langage C. Tous les identificateurs d'attribut doivent être précédés du caractère "`$`".

Chaque non-terminal de la grammaire qui utilise un attribut doit être préalablement déclaré dans la zone de déclaration des attributs.

Après chaque règle de grammaire, il faut toujours déclarer les attributs utilisés par le non-terminal qui constitue la partie gauche de la règle de grammaire.

4 Déclaration des attributs synthétisés

La déclaration des attributs se décompose en deux parties

1. Avant l'utilisation des attributs, le programmeur doit les déclarer globalement en explicitant les non-terminaux qui les utilisent ainsi que le type C de l'attribut.

   Cette déclaration doit toujours précéder les règles de la grammaire.

   La grammaire syntaxique des déclarations d'attibuts est :

   ```
   <declaration-globale> = <declaration-attributs> "$"
   ```

   Le signe "`$`" qui termine la déclaration doit forcément se trouver sur la première colonne d'une nouvelle ligne.

   ```
   <declaration-attributs> = <declaration-attribut>

   <declaration-attributs> = <declaration-attribut>
                             <declaration-attributs>

   <declaration-attribut> = "$" <identificateur-C>
                            "(" <non-terminal-lotos>
                            {"," <non-terminal-lotos>} ")"
                            ":" <type-C> ";"
   ```

   Le signe "`$`" doit toujours se trouver sur la première colonne d'une nouvelle ligne. `<identificateur-C>` et `<type-C>` correspondent à des identificateurs de variable et de type respectant la syntaxe du langage C.

   On déclare globalement l'attribut compteur comme un entier. Cet attribut est associé au non-terminal `<identificateur>`.

```
$compteur(<identificateur>):int;
```

2. Après chaque règle de la grammaire syntaxique, le programmeur doit déclarer localement les attributs utilisés par le non-terminal qui forme la partie gauche de la règle.

   La grammaire syntaxique de cette déclaration locale est :

   ```
   <declaration-locale> = <declaration-attribut-ligne>
                          {<declaration-attribut-ligne>}

   <declaration-attribut-ligne> = <declaration-attribut>
                                  {"," <declaration-attribut>}

   <declaration-attribut> = "$" <identificateur-C>
                            "(" <non-terminal-lotos> ")"
   ```

   `<non-terminal-lotos>` doit toujours être le non-terminal de la partie gauche de la règle de la grammaire.

   De plus, à chaque ligne de déclaration locale d'attributs le signe "`$`" du premier attribut doit être sur la première colonne.

   Soit la règle :

   ```
   <identificateur> = <suite-de-lettre> ;
                      $compteur(<identificateur>)
   ```

   Cette règle déclare localement l'attribut compteur associé au non-terminal `<identificateur>`.

5 Description des actions synthétisées

Les actions synthétisées sont toujours décrites après chaque règle de grammaire à la suite de la déclaration locale d'attributs.

La description des actions ne doit jamais commencer sur la première colonne de chaque ligne.

Pour utiliser un attribut dans la description d'une action synthétisée, il suffit de taper :

```
"$"<identificasteur-C> "("<non-terminal>")"
```

où `<non-terminal>` désigne un non-terminal de la règle de la grammaire pour laquelle on écrit l'action. L'attribut considéré est alors celui associé à ce non-terminal.

Si il existe plusieurs non-terminaux identiques dans la règle de la grammaire, la distinction des attributs s'effectue en ajoutant un ou plusieurs caractères `'` après le `<non-terminal>` et la parenthèse "`)`".

soit la règle :

```
<liste> = <liste> <idf> <liste> ;
```

et l'attribut associe exp. `$exp(<liste>)` désigne l'attribut associé au non-terminal composant la partie gauche de la règle.

`$exp(<liste>')` désigne l'attribut associé au non-terminal `<liste>` précédant le non-terminal `<idf>`.

`$exp(<liste>")` désigne l'attribut associé au non-terminal `<liste>` suivant le non-terminal `<idf>`.

Soit une grammaire définissant des listes composées par les mots `idf` séparés par une virgule. On désire compter le nombre d'éléments composant cette liste.

Pour cela, on utilise un attribut entier servant de compteur qui représente le nombre d'éléments de chaque liste.

Le fichier *mots.semc* correspondant s'écrit alors :

```
     *declaration locale des attributs
     $compteur(<axiom>, <liste>):int;
     *une declaration se termine toujours par le signe $
     $

     *Description des regles de la grammaire et des actions
     *associees

     <axiome> = <liste> ;
     *declaration locale
     $compteur(<axiome>)
     *action associee a cette regle
 $compteur(<axiome>) = $compteur(<liste>);
 printf("nombre d'elements dans la liste %d",$compteur(<axiome>);

     <liste> = <identificateur> ;
     *declaration locale
     $compteur(<liste>)
     *action associee a cette regle
$compteur(<liste>) = 1;

     <liste> = <liste> , <identificateur> ;
     *declaration locale
     $compteur(<liste>)
     *action associee a cette regle
$compteur(<liste>) = 1 + $compteur(<liste>');

     <identificateur> = "idf" ;
```

6 Compléments sur l'analyse syntaxique

Le programmeur, après la déclaration globale des attributs peut déclarer des variables C, inclure des fichiers de suffixe *.h*, appeler des actions (toujours

décrites en langage C) qui sont effectuées au début de l'analyse syntaxique, ou à la fin de l'analyse.

Ces déclarations doivent s'effectuer de la manière suivante :

```
decl=
* zone de declaration
* inserer ici la declaration de variable ou les inclusions de
* fichiers
$
open=
* actions a effectuer a l'initialisation de l'analyse
* syntaxique
$
close=
* actions a effectuer a la fin de l'analyse syntaxique
$
```

Il est à noter que si SYNTAX n'arrive pas à rattraper une erreur de syntaxe, il continue l'analyse syntaxique, mais n'exécute plus les actions lors de la réduction des règles.