Sémantique et TDL.
Génération de code

Considérons la grammaire décrivant les instructions du langage Bloc :

1. $S \rightarrow B$
2. $B \rightarrow \{\ LI\ \}$
3. $LI \rightarrow I\ LI$
4. $LI \rightarrow \Lambda$
5. $I \rightarrow \textbf{const}\ T\ \textbf{id} = V\ ;$
6. $I \rightarrow T\ NI = E\ ;$
7. $I \rightarrow A = E\ ;$
8. $I \rightarrow \textbf{if}\ (\ E\ )\ B\ \textbf{else}\ B$
9. $I \rightarrow \textbf{if}\ (\ E\ )\ B$
10. $I \rightarrow \textbf{while}\ (\ E\ )\ B$
11. $I \rightarrow \textbf{print}\ E\ ;$
12. $T \rightarrow \textbf{int}$
13. $T \rightarrow \textbf{boolean}$
14. $T \rightarrow <\ T\ ,\ T\ >$
15. $E \rightarrow \ldots$
16. $I \rightarrow \textbf{typedef}\ T\ \textbf{id}\ ;$
17. $T \rightarrow \textbf{id}$
18. $T \rightarrow \textbf{struct id}\ \{\ LC\ \}$
19. $LC \rightarrow C\ LC$
20. $LC \rightarrow \Lambda$
21. $C \rightarrow T\ \textbf{id}\ ;$
22. $E \rightarrow E\ .\ \textbf{id}$
23. $E \rightarrow \{\ LE\ \}$
24. $LE \rightarrow E\ ,\ LE$
25. $LE \rightarrow E$
30. $NI \rightarrow NI\ [\ ]$
31. $NI \rightarrow (\ NI\ )$
32. $NI \rightarrow *\ NI$
33. $NI \rightarrow \textbf{id}$
34. $E \rightarrow E\ [\ E\ ]$
35. $E \rightarrow *\ E$
36. $E \rightarrow \&\ E$
37. $E \rightarrow \textbf{new}\ T\ [\ E\ ]$
38. $A \rightarrow A\ [\ E\ ]$
39. $A \rightarrow (\ A\ )$
40. $A \rightarrow *\ A$
41. $A \rightarrow A\ .\ \textbf{id}$
42. $A \rightarrow A\ \textbf{id}$

1 Types simples et couple

Soit le programme :

```
pgcd {
  <int,int> c = {47,53};
  const int test = 0;
  int a = fst c;
  int b = snd c;
  while (a * b != test) {
    if ( a > b ) {
      int na = a-b;
      a = na;
    } else {
      int nb = b-a;
      b = nb;
    }
  }
  int res = a;
  if (res == test) {
    res = b;
  }
  print res ;
}
```

1. Traduire ce programme en langage assembleur pour la machine virtuelle TAM en utilisant la position des identificateurs calculées dans la séance précédente.

2. Proposer des actions sémantiques pour la génération de code. Ces actions se concrétisent sous la forme de méthodes dans les classes JAVA constituant l'arbre abstrait.

2 Type enregistrement

Soit le programme :

```
test {
   typedef struct Pointi { int x; int y;} Point;
   typedef struct Segmenti {Point ext1; Point ext2;} Segment;
   Segment s = {{0,1},{2,3}};
   int x1 = s.ext1.x;
   int y2 = s.ext2.y;
   s.ext2.x = x1;
   s.ext1.y = y2;
}
```

1. Calculer la position des identificateurs.

2. Traduire ce programme en langage assembleur pour la machine virtuelle TAM.

3. Proposer des actions sémantiques pour la génération de code. Ces actions se concrétisent sous la forme de méthodes dans les classes JAVA constituant l'arbre abstrait.

3 Type tableau et pointeur

Soit l'exemple :

```
test {
   int v = 1;
   int *ptr = &v;
   int j = *ptr;
   *ptr = 2;
   int t[] = new int[5];
   int i = t[3];
   t[3] = 4;
}
```

1. Calculer la position des identificateurs.

2. Traduire ce programme en langage assembleur pour la machine virtuelle TAM.

3. Proposer des actions sémantiques pour la génération de code. Ces actions se concrétisent sous la forme de méthodes dans les classes JAVA constituant l'arbre abstrait.