Sémantique et TDL.
Typage et Gestion mémoire

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. $I \rightarrow T\ \textbf{id} = E\ ;$
7. $I \rightarrow \textbf{id} = E\ ;$
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. $LE \rightarrow E$

Le diagramme de classe (simplifié) de l'AST est également rappelé.

1 Type tableau et pointeur

Nous étendons le langage Bloc par la notion de type tableau et de type pointeur. Nous nous limitons aux modifications de la grammaire liées à la déclaration et à l'affectation de variables. Des modifications similaires doivent être effectuées au niveau de la déclaration de type et de champs dans un enregistrement.

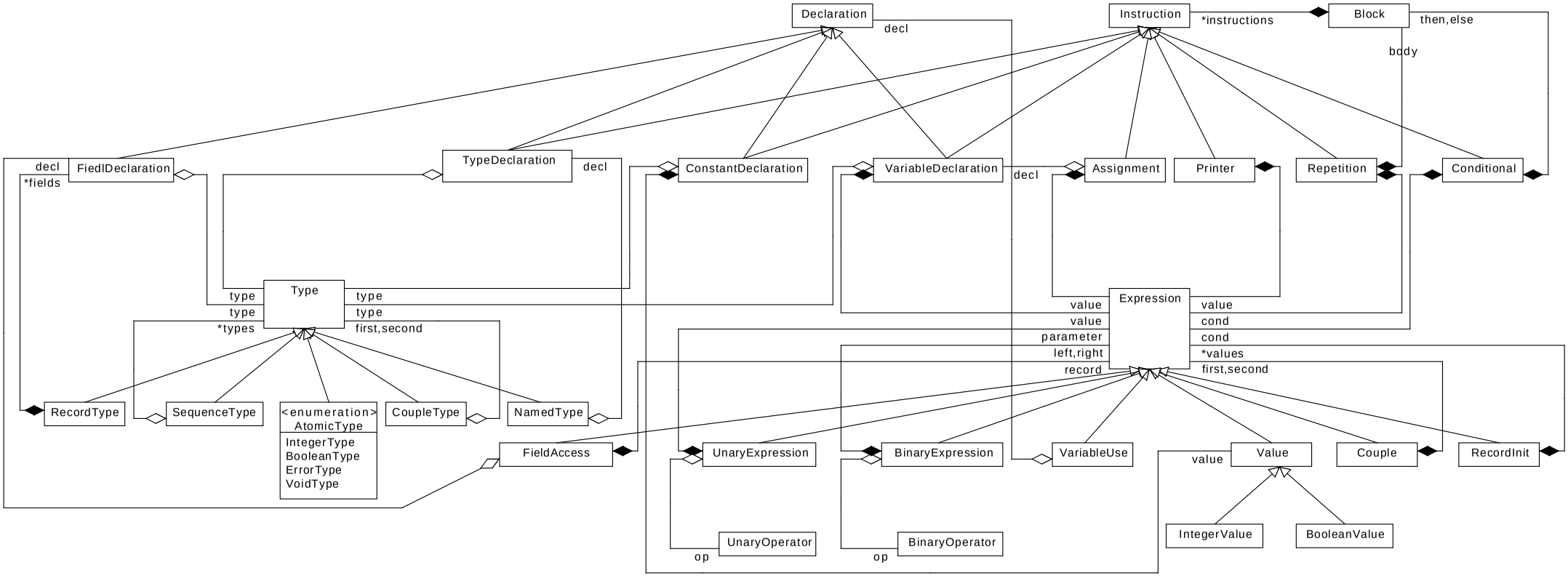
1.1 Déclaration de variable

La grammaire est modifiée de la façon suivante :

6. $I \rightarrow T\ NI = E\ ;$
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. $E \rightarrow \&\ \textbf{id}$
37. $E \rightarrow \textbf{new}\ T\ [\ E\ ]$
38. $E \rightarrow \textbf{new T}$

Soit un exemple associé :

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


Declaration
decl
Instruction
*instructions
Block
then,else
body
decl
*fields
FiedlDeclaration
TypeDeclaration
decl
ConstantDeclaration
VariableDeclaration
decl
Assignment
Printer
Repetition
Conditional
type
type
*types
Type
type
type
first,second
value
value
parameter
left,right
record
Expression
value
cond
cond
*values
first,second
RecordType
SequenceType
<enumeration>
AtomicType
IntegerType
BooleanType
ErrorType
VoidType
CoupleType
NamedType
FieldAccess
UnaryExpression
BinaryExpression
VariableUse
value
Value
Couple
RecordInit
UnaryOperator
op
BinaryOperator
op
IntegerValue
BooleanValue

1. Donner le type de chaque variable.
2. Compléter le diagramme de classe de l'AST pour ajouter la ou les classes nécessaires à la gestion des tableaux et pointeurs.
3. Définir les attributs et actions sémantiques nécessaires pour construire l'arbre abstrait correspondant aux types et expressions tableau et pointeur.
4. Définir le traitement sémantique nécessaire pour la résolution des identifiants d'un programme en langage Bloc ainsi modifié.
5. Définir le traitement sémantique nécessaire pour vérifier qu'un programme en langage Bloc ainsi modifié est bien typé.

1.2 Affectation de variable

L'instruction d'affectation est modifiée pour que la partie gauche puisse manipuler des tableaux, pointeurs et enregistrements.

7. $I \rightarrow A = E$;
38. $A \rightarrow A$ [E]
39. $A \rightarrow$ (A)
40. $A \rightarrow * A$
41. $A \rightarrow A$. **id**
42. $A \rightarrow$ **id**
42. $A \rightarrow$ **id**

Soit un exemple associé :

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

1. Compléter le diagramme de classe de l'AST pour ajouter la ou les classes nécessaires à la gestion des affectation des tableaux et pointeurs.
2. Définir les attributs et actions sémantiques nécessaires pour construire l'arbre abstrait correspondant aux affectations d'expressions tableau et pointeur.
3. Définir le traitement sémantique nécessaire pour la résolution des identifiants d'un programme en langage Bloc ainsi modifié.
4. Définir le traitement sémantique nécessaire pour vérifier qu'un programme en langage Bloc ainsi modifié est bien typé.

2 Gestion mémoire pour types simples et les couples

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 ;
}
```

Nous avons présenté en cours la machine virtuelle TAM que nous utiliserons comme cible de la génération de code. Il s'agit d'une machine à pile, c'est-à-dire que toutes les données manipulées seront stockées dans le pile pour les allocations statiques (variables, paramètres des fonctions) et dans le tas pour les allocations dynamiques.

Dans cette première partie, nous n'utiliserons que la pile.

1. Réprésenter l'allocation des variables dans la pile au fur et à mesure de l'exécution du programme précédent.

2. Proposer des actions sémantiques pour associer à chaque déclaration de variable sa position dans la pile. Ces actions se concrétisent sous la forme de méthodes dans les classes Java constituant l'arbre abstrait.

3 Cas des types enregistrements, pointeurs et tableaux

1. Proposer des évolutions de la solution précédente pour prendre en compte les types enregistrements, pointeurs et tableaux.