

Exécution de l'arbre abstrait

Alexis Nasr
Franck Dary
Pacôme Perrotin

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Exécution de l'arbre abstrait

- L'arbre abstrait est une représentation structurée du programme
- Il est possible d'exécuter les instructions de l'arbre abstrait sur des données pour fournir un résultat
- Les instructions sont exécutées par le langage hôte (le langage dans lequel est écrit l'interpréteur)
- L'interprétation est réalisée en parcourant l'arbre abstrait
- Ce parcours n'est plus un parcours en profondeur systématique
- L'ordre de parcours est altéré pour :
 - Les boucles (tantque)
 - Les conditionnelles (si alors sinon)
 - Les appels de fonction

Cas simple : les expressions

```
public Integer visit(SaExpAdd node)
{
    int op1 = node.getOp1().accept(this);
    int op2 = node.getOp2().accept(this);
    return op1 + op2;
}
```

- Cela correspond à un parcours descendant de l'arbre
- Les valeurs calculées remontent dans l'arbre
- Comme des attributs synthétisés!

Conditionnelles

```
public Integer visit(SaInstSi node)
{
    int test = node.getTest().accept(this);
    if(test != 0)
        node.getAlors().accept(this);
    else
        if(node.getSinon() != null)
            node.getSinon().accept(this);
    return 1;
}
```

- L'expression correspondant au test est interprétée
- Le résultat est affecté à la variable test
- Si test est différent de 0, on effectue le bloc d'instruction alors
- Sinon, on effectue le bloc d'instruction sinon

Boucles

```
public Integer visit(SaInstTantQue node)
{
    int test = node.getTest().accept(this);
    while (test != 0){
        node.getFaire().accept(this);
        test = node.getTest().accept(this);
    }
    return 1;
}
```

- L'expression correspondant au test est interprétée
- Le résultat est affecté à la variable test
- Tant que test est différent de 0
- On interprète le bloc d'instruction faire
- On interprète l'expression test

Variables globales

```
public Integer visit(SaVarSimple node)
{    // variable globale
    if(node.tsItem.portee == this.tableGlobale){
        val = varGlob[node.tsItem.adresse];
    }
}
```

- Elles sont stockées dans un tableau varGlob
- On accède à une variable grâce à son adresse, stockée dans la table des symboles globale

Variables locales et paramètres

```
public class SaEnvironment {  
    private int[] vars;  
    private int[] args;  
    private int returnValue;  
}
```

- Pour chaque appel de fonction, de la mémoire doit être allouée pour y stocker
 - Les variables locales
 - Les paramètres
 - La valeur de retour
- Toutes ces informations sont stockées dans une instance de la classe SaEnvironment

Accès aux variables

```
public Integer visit(SaVarSimple node){  
    int val = 0;  
    // variable globale  
    if(node.tsItem.portee == this.tableGlobale){  
        val = varGlob[node.tsItem.adresse];}  
    else if(node.tsItem.isParam){ // parametre  
        val = curEnv.getArg(node.tsItem.adresse);}  
    else { // variable locale  
        val = curEnv.getVar(node.tsItem.adresse);}  
    return val;  
}
```

- La variable curEnv est l'environnement correspondant à la fonction en train d'être interprétée.

Appel de fonction

- Comment simuler un appel de fonction d'une fonction qui n'a pas été écrite dans le langage hôte?
- Au lieu de faire $f(x)$
- On fait `appel(f, x)`

Appel de fonction

- Création d'un nouvel environnement
- Calcul de la valeur des paramètres, stockage dans l'environnement
- Sauvegarde de l'environnement courant
- Le nouvel environnement devient l'environnement courant
- Interprétation du corps de la fonction
- Récupération de la valeur de retour dans l'environnement
- Restauration de l'environnement
- Retour de la valeur de retour

Appel de fonction

```
public Integer visit(SaAppel node){
    TsItemFct fct = node.tsItem;
    int i = 0;
    SaEnvironment newEnv = new SaEnvironment(node.tsItem);
    for(SaExp lArgs = node.getArguments(); lArgs != null; lArgs =
        newEnv.setArg(i++, lArgs.getTete().accept(this));
        //sauvegarde de l'env courant pour le restaurer après l'appel
    SaEnvironment oldEnv = curEnv;
    // le nouvel env devient l'env courant
    curEnv = newEnv;
    // on exécute le corps de la fonction
    fct.saDecFonc.getCorps().accept(this);
    int returnValue = curEnv.getReturnValue();
    //restauration de l'env d'avant appel
    curEnv = oldEnv;
    return returnValue;}
```

La classe SaEval

```
public class SaEval extends SaDepthFirstVisitor <Integer> {  
    private Ts tableGlobale;  
    private SaEnvironment curEnv;  
    private int[] varGlob;  
  
    public SaEval(SaNode root, Ts tableGlobale){  
        this.tableGlobale = tableGlobale;  
        curEnv = null;  
        varGlob = new int[tableGlobale.nbVar()];  
  
        SaAppel appelMain = new SaAppel("main", null);  
        appelMain.tsItem = tableGlobale.getFct("main");  
  
        appelMain.accept(this);  
    }  
}
```

La machine virtuelle SaVM

```
public class SaVM {  
  
    public static void main(String[] args) {  
        ...  
        DOMParser parser = new DOMParser();  
        parser.parse(saFileName);  
        Document document = parser.getDocument();  
        Node rootNode=document.getDocumentElement();  
        SaProg prog = processSaProg(rootNode);  
        Ts tableGlobale = new Sa2ts(prog).getTableGlobale();  
        SaEval saEval = new SaEval(prog, tableGlobale);  
    }  
}
```

Utilisation de la machine virtuelle

```
$ java Compiler add1.l -v 2
$ ls add1.*
add1.l  add1.sa  add1.sc  add1.ts
$ java -cp "....xerces-2_12_1/*" SaVM -sa add1.sa
13
```