Rapport de Compilation: VSL+ ANTLR/Java

Auteurs : HOUSSEIN GALIB, Waberi - LEDOUX, Simon
Encadrant de TP :
MASSON, Véronique
Master 1, Génie logiciel, Groupe 1.2

Introduction

Dans le cadre du module COMPILATION, nous avons réalisé un compilation du langage VSL+ à l'aide de ANTLR qui produit du code trois adresses. Nous avons travaillé dans trois fichiers principaux afin d'ajouter les différentes constructions du langage VSL+ sur ce qui nous a été donné initialement. Pour cela nous avons décrit un arpenteur d'arbre qui va effectuer la vérification de type et la génération de code trois adresses pour chaque constructions du langage VSL+. Dans un premier temps nous avons constitué la face avant de notre compilateur et puis en seconde nous avons transformé le code de trois adresses en un fichier assembleur MIPS afin de produire des exécutables.

1 Méthodologie de travail

Après avoir analysé les différents classes fournies avec le projet, nous avons testé les expressions simples qui était déjà compilable et utilisable.
Nous avons commencé avec un simple fichier de code test en VSL+ contenant des expressions simples. Puis nous avons ensuite implémenté les fonctionnalités du langage une à une c'est à dire de façon itérative. Dans la suite nous complétons notre fichier VSL+ de test pour tester les fonctionnalités qui viennent d'être ajoutées. Quand on met à jour notre code VSL+ de test, nous gardons les anciens tests créés pour les fonctionnalités rajoutées auparavant.

Modifications apportées:

– VSLTreeParser.g: la vérifications des types s'est fait à ce niveau, et la modification de l'axiome pour pouvoir faire des tests.

– Code3aGenerator.java: Ajout des méthodes du code générateur de 3 adresses pour les différentes instructions de VSL+.

– TypeCheck.java: Ajout une méthode pour la vérification de type unaire de VSL+.

2 Bilan du travail réalisé

Nous avons pu réaliser les constructions suivantes avec leurs tests fonctionnels :

– Les expressions simples
– L'instruction d'affectation
– La déclaration des variables
– Les expressions avec variables
– La gestion des blocs
– Les instructions de contrôle if , while et la séquence
– La définition et l'appel de fonctions (avec les prototypes)
– Les fonctions de la bibliothèque : PRINT et READ
– La gestion des tableaux (déclaration, expression, affectation et lecture)

On a pas pu traiter la gestion des tableaux(déclaration, expression, affectation et lecture) et tous les cas d'erreurs que peut générer un programme VSL+, comme le contrôle du nombre de paramètres d'une fonction déjà prototypée.
Les tests réalisés sur ces constructions prouvent que notre compilateur est performant dans son domaine de définition. Nous sommes satisfaits d'arriver à une telle étape de notre compilateur, car au début c'était difficile de comprendre ce qui se passait quand on compilait un programme.

3 programme VSL+

Exemple de compilation d'un programme VSL+

PROTO INT fact(k)
FUNC VOID main()
{
INT n, i, t[11]
PRINT "Input n between 0 and 11:\n"
READ n
i := 0
WHILE n-i
DO
{
/* t[i] := fact(i) */
i := i+1
}
DONE
i := 0
WHILE n-i
DO
	Production du code 3 adresses
	label main
	beginfunc
	var n
	var i
	var t
	arg L10
	call L4
	n = call L8
	i = 0
	label L11
	var T_0
	T_0 = n - i
	ifz T_0 goto L12
	var T_1

```
{
/* PRINT "f(", i, ") = ", t[i], "\n" */
i := i+1
}
DONE
}
FUNC INT fact(n)
{
IF n THEN RETURN n*fact(n-1)
ELSE RETURN 1
FI
}
```

```
T_1 = i + 1
i = T_1
goto L11
label L12
i = 0
label L13
var T_2
T_2 = n - i
ifz T_2 goto L14
var T_3
T_3 = i + 1
i = T_3
goto L13
label L14
endfunc
label fact
beginfunc
var n
ifz n goto L15
var T_5
var T_4
T_4 = n - 1
arg T_4
T_5 =  call fact
var T_6
T_6 = n * T_5
return T_6
goto L16
label L15
return 1
label L16
endfunc
L10:"Input n between 0 and 11:\n"
```

– Production du code assembleur MIPS

```
        .rdata
        .align 2
        $LC0:
                .ascii    "%d\000"
                .align 2
        L10:
                .ascii      "Input n between 0
and 11:\012\000"
        .text
        # label main
        # beginfunc
           .align    2
           .globl main
           .ent main
        main:
            addiu    $sp,$sp,-84
            sw    $31,16($29)
            sw    $16,24($29)
            sw    $17,28($29)
            sw    $18,32($29)
            sw    $19,36($29)
            sw    $20,40($29)
            sw    $21,44($29)
            sw    $22,48($29)
            sw    $23,52($29)
        # var n
        # var i
        # var t
        # arg L10
                lui   $8,%hi(L10)
                addiu   $8,$8,%lo(L10)
            sw   $8,0($29)
           move   $4,$8
        # call L4
            jal   n_printf
            nop
        # n =  call L8
                addiu   $5,$29,56
        # label fact
        # beginfunc
           .align    2
           .globl fact
           .ent fact
        fact:
            addiu    $sp,$sp,-36
            sw    $31,16($29)
        # var n
        # ifz n goto L15
                lw  $8,36($29)
           beq  $8,0,L15
        # var T_5
        # var T_4
        # T_4 = n - 1
                li   $9, 1
           sub $9, $8, $9
        # arg T_4
                sw  $9,28($29)
            sw   $9,0($29)
           move   $4,$9
        # T_5 =  call fact
            jal   fact
            nop
        # var T_6
        # T_6 = n * T_5
                lw  $9,36($29)
                move  $8,$2
           mult $9, $8
           mflo $8
        # return T_6
                sw  $2,24($29)
                move  $2,$8
            lw    $31,16($29)
            addiu    $sp,$sp,36
            jr  $31
            nop
        # goto L16
```

```
        lui   $4,%hi($LC0)
        addiu   $4,$4,%lo($LC0)
    jal   n_read_int
    nop
        sw   $2,56($29)
# i = 0
        li   $8, 0
# label L11
        sw  $2,56($29)
        sw  $8,60($29)
L11:
# var T_0
# T_0 = n - i
        lw  $9,56($29)
        lw  $8,60($29)
   sub $8, $9, $8
# ifz T_0 goto L12
        sw  $8,68($29)
   beq  $8,0,L12
# var T_1
# T_1 = i + 1
        lw  $11,60($29)
        li   $10, 1
   add $10, $11, $10
# i = T_1
        sw  $10,72($29)
# goto L11
        sw  $10,60($29)
   j  L11
# label L12
L12:
# i = 0
        li   $8, 0
# label L13
        sw  $8,60($29)
L13:
# var T_2
# T_2 = n - i
        lw  $9,56($29)
        lw  $8,60($29)
   sub $8, $9, $8
# ifz T_2 goto L14
        sw  $8,76($29)
   beq  $8,0,L14
# var T_3
        sw  $2,32($29)
   j  L16
# label L15
L15:
# return 1
        li   $2, 1
    lw    $31,16($29)
    addiu    $sp,$sp,36
    jr  $31
    nop
# label L16
L16:
# endfunc
    lw    $31,16($29)
    addiu    $sp,$sp,36
    jr  $31
    nop
   .end fact
```

```
# T_3 = i + 1
        lw  $11,60($29)
        li   $10, 1
   add $10, $11, $10
# i = T_3
        sw  $10,80($29)
# goto L13
        sw  $10,60($29)
   j  L13
# label L14
L14:
# endfunc
    lw    $16,20($29)
    lw    $17,24($29)
    lw    $18,28($29)
    lw    $19,32($29)
    lw    $20,36($29)
    lw    $21,40($29)
    lw    $22,44($29)
    lw    $23,48($29)
    lw    $31,16($29)
    addiu    $sp,$sp,84
    jr  $31
    nop
   .end main
```

Autre exemple avec MIPS:

```
FUNC VOID main()
{
  INT i,j,k,l
  i := 0
  j := 0
  k := 0
  l := 0
  {
        l := 1
        {
        k:= 2
        {
        j:= 3
        }
```

```
label main
beginfunc
var i
var j
var k
var l
i = 0
j = 0
k = 0
l = 0
l = 1
k = 2
j = 3
```

```
        }
    }
     PRINT "Et voila: ", i,j,k,l
}
```

```
arg L10
call L4
arg i
call L2
arg j
call L2
arg k
call L2
arg l
call L2
endfunc
L10:"Et voila: "
```

Code assembleur MIPS

```
.rdata
.align 2
$LC0:
        .ascii    "%d\000"
        .align 2
L10:
        .ascii    "Et voila: \000"
.text
# label main
# beginfunc
   .align    2
   .globl main
   .ent main
main:
    addiu    $sp,$sp,-72
    sw    $31,16($29)
    sw    $16,24($29)
    sw    $17,28($29)
    sw    $18,32($29)
    sw    $19,36($29)
    sw    $20,40($29)
    sw    $21,44($29)
    sw    $22,48($29)
    sw    $23,52($29)
# var i
# var j
# var k
# var l
```

```
        move $5,$4
        lui   $4,%hi($LC0)
        addiu   $4,$4,%lo($LC0)
    jal   n_printf
    nop
# arg j
        lw  $8,60($29)
    sw   $8,0($29)
   move   $4,$8
# call L2
        move $5,$4
        lui   $4,%hi($LC0)
        addiu   $4,$4,%lo($LC0)
    jal   n_printf
    nop
# arg k
        lw  $8,64($29)
    sw   $8,0($29)
   move   $4,$8
# call L2
        move $5,$4
        lui   $4,%hi($LC0)
        addiu   $4,$4,%lo($LC0)
    jal   n_printf
    nop
# arg l
        lw  $8,68($29)
    sw   $8,0($29)
```

```
# i = 0
        li   $8, 0
# j = 0
        li   $9, 0
# k = 0
        li   $10, 0
# l = 0
        li   $11, 0
# l = 1
        li   $12, 1
# k = 2
        li   $11, 2
# j = 3
        li   $10, 3
# arg L10
        lui   $9,%hi(L10)
        addiu   $9,$9,%lo(L10)
    sw   $9,0($29)
   move   $4,$9
```

```
   move   $4,$8
# call L2
        move $5,$4
        lui   $4,%hi($LC0)
        addiu   $4,$4,%lo($LC0)
    jal   n_printf
    nop
# endfunc
    lw    $16,20($29)
    lw    $17,24($29)
    lw    $18,28($29)
    lw    $19,32($29)
    lw    $20,36($29)
    lw    $21,40($29)
    lw    $22,44($29)
    lw    $23,48($29)
    lw    $31,16($29)
    addiu    $sp,$sp,72
    jr  $31
    nop
   .end main
```

Conclusion

Ce travail nous a permis de comprendre les techniques utilisées en compilation, les fonctionnement d'un compilateur de la production de code compilable à l'exécution. Notre compilateur n'étant pas réalisé en intégralité car nous n'avons pas pu réaliser quelques constructions comme les tableaux et la gestion d'erreur. Ce qui nous a pris beaucoup du temps pour les traiter et enfin nous avons rendu un code fonctionnel.