Nicolas Demongeot

Jean Derieux

Rapport Projet

Programmation dirigée par la syntaxe

Table des matières

Le But du TP :

Le but du TP est de construire un traducteur de langage vsl+ vers un code intermédiaire LLVM .

Pour traduire le code vsl+ vers LLVM on va utiliser ANTLR qui va nous permettre de générer automatiquement à partir d'un stream en entrée , une instance du programme associé.

On va d'abord implémenter une grammaire du langage vsl+ en y intégrant du code java pour chaque règle de la grammaire. Ce code java correspond aux traitements à effectuer pour créer le type associé en java. Nous allons confier notre grammaire à notre ami ANTLR qui lui va, pour une instance donnée d'un texte représentant un programme vsl+, nous créer un Objet Program.

Avec cet Objet Program on va pouvoir appeler la méthode ToIr() qui va générer le code IR associé. Le déclenchement de la méthode ToIr() de programme va provoquer l'exécution de cette même méthode sur les sous objets c'est a dire sous-types composant notre programme. Construisant le code de chaque instruction et l'ajoutant a la pile jusqu'à obtenir un code intermédiaire complet et ordonné que nous n'aurons plus qu'à imprimer sur une sortie standard.

Il est aussi possible d'obtenir ne version pretty-printé du programme d'entrée via l'instruction (Program) prog.pp() .

État courant du projet :

Nous avons traité la majeure partie du projet .

Nous avons couvert les expressions simple,les variables, les conditionnelles ,les boucles et les fonctions. Toutefois nous n'avons pas eu le temps d'aller plus loin dû aux divers travaux en cours actuellement bien que nous ayons quelques idée sur la manière d'implémenter la partie tableaux, du moins la déclaration de celui-ci.

1) Passage Du Vsl au code Intermédiaire

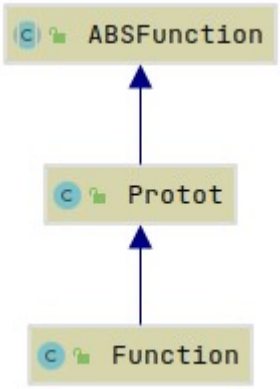
Prérequis : Le programme d'entrée doit être accepter par le parser que nous avons créé, c'est à dire respecter les règles de constructions du langage.

Ensuite Le schéma de base pour passer de l'entrée à la sortie se passe en 3 temps :

-Le programme est parser et en ressort un Objet Programme comme expliqué précédemment,

Un programme étant défini comme un ensemble de fonctions et de prototype, Un objet Program dispose d'une liste d'ABSfunction. Une ABSfunction est une classe abstraite qui peut être soit une fonction ou un prototype.

Pour produire un objet IR on va appeler la fonction IR sur tout les ABSfunction.



1.1 Comment étendre l'ASD :

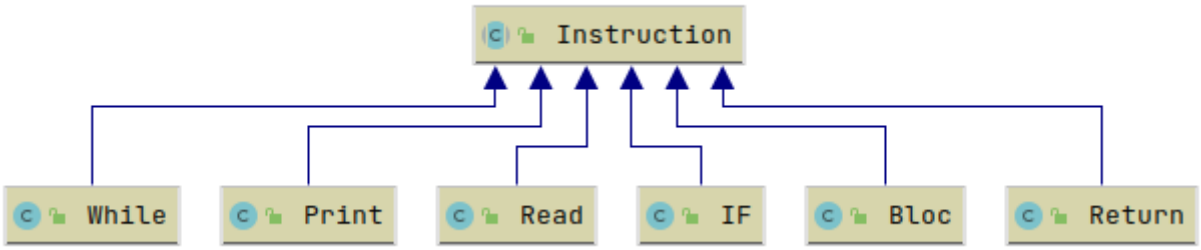
Exemple de production d'un code Llvm.IR pour une Instruction:

Pour étendre l'ASD par exemple en ajouter un nouveau type d'instruction il va falloir ajouter une règle dans la grammaire (ie le parser) au niveau de instruction si cette instruction nécessite l'utilisation de nouveau mot clef comme IF THEN ELSE FI par exemple il faudra les ajouter au lexer.

Ensuite on va ajouter le code java qui doit s'exécuter pour produire en sortie un TP2.ASD.Instruction.

Par exemple pour créer une instruction if on va avoir besoin de deux instructions correspondant a l'instruction devant s'exécuter au niveau du then si la condition est évalué a true et celle au niveau du ELSE et d'un expression pour évaluer la condition.

On va créer une classe qui va extends TP2.ASD.Instruction pour profiter de l'injection de dépendance.



Ici la classe « IF », lorsque la règle correspondante va être atteinte dans le parser on retournera un new IF($cond.out, $expression1.out, $expression2.out) ;

il va falloir implémenter les deux méthodes de la classe abstraite instruction.

Méthode ToIR() :

Pour cette méthode on va utiliser une instruction de comparaison icmp,

et deux instructions de branchements br.

On va créer une sous-classe qui extends Llvm.Instruction pour chacune de ses instructions.

Une fois que l'on a tout le matériel a notre disposition on va (enfin) pouvoir implémenter la méthode IR().

Ici on va d'abord « évalue r » la valeur de la condition avec la méthode TOIR()

pour récupérer son adresse mémoire et son type pour des vérification de type.

On va crée une instruction icmp avec l'adresse résultat de la condition.

On instancie les instructions de branchement br.

On les ajoutes dans l'ordre souhaiter de flot d'exécution.

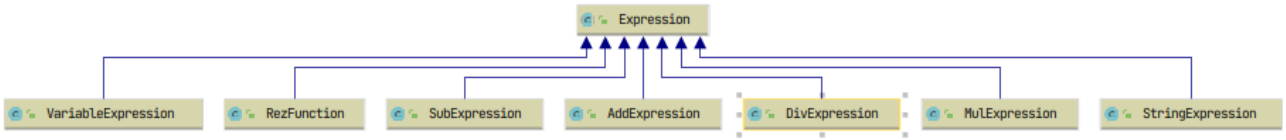
On ajoute dans le bon ordre dans le l'instruction de retour contenant le code ir.

2)Vérification et contrôle

Par quels moyens effectuons nous des contrôles ?

On va effectuer la majorité des contrôles grâce à l'héritage d'une table de symbole passer a tout les types lors de l'appel de la méthode ToIR().

2 .1 Gestion des expressions



Pour les expressions on va effectuer des contrôle de type lors d'opération binaire comme l'addition, la soustraction,multiplication etc.

-on va vérifier que le type de l'expression de gauche est bien de même type que celle de droite sinon on throw un nouveau Type exception.

-Dans le cas d'une variable expression si on ne trouve pas le symbole correspondant dans la table de symbole on throw un UndeclaredVariableexception.

-Pour RezFunction qui est le résultat d'un appel à une function, si la fonction n'est pas dans la tablesymbole on throw une exception si la fonction est dans la table symbole mais est appeler avec un mauvais nombre d'argument on throw un WrongParamFunctionCall().

2.2 Contrôle Affectation :

Vérification de type si l'expression à affecter n'est pas du bon type throw type exception

si pas la variable n'est pas déclarer throw type exception aussi.

2.3 Contrôle Déclaration :

-Déclaration de prototype si le nom est déjà dans la table symbole on throw un VariableAlreadydeclared().

-Declaration d'un entier si le nom est déjà dans le stable symbole on déclare un DeclarationException().

3 Choix implémentation

3.1 If Then et If Then Else :

```
instruction returns [TP2.ASD.Instruction out]

IF cond = expression THEN {boolean a = false;}
        (i=instruction)
        (ELSE (i1=instruction){a=true;})?
        FI
        {if(a){
           $out=new TP2.ASD.IF($cond.out,$i.out,$i1.out);
        }else{
        $out=new TP2.ASD.IF($cond.out,$i.out,null);}
        }
```

Dans le parser il n'existe qu'une seule règle concernant ses deux structures si on n'a pas de Else on passera un null en troisième argument.

3.2 sur les fonctions

-Le contenus des fonctions sont ajouter a l'endroit de la définitions de la fonction même si un prototype l'a définis plus haut.

3.3 sur les déclarations

On peut déclarer plusieurs fois une variable avec un même nom si elles sont sur des blocs différents c'est a dire avec des portées différentes.

Lors de la recherche d'un variable on prendra celle de portée moindre.