Rapport PDS TP2

par Guerin Alexys

Introduction :

Dans le cadre du cours de Programmation Dirigée par la Syntaxe, notre second travail pratique, nous devions réaliser un compilateur du langage VSL+ à l'aide de Ocaml ou Java(avec Llvm). Notre travail consistait à réaliser la génération de l'AST(arbre de syntaxe abstraite) à partir d'un fichier source VSL+, puis réaliser une vérification de type et la génération du code à partir de notre AST ainsi que produire du code 3 adresses en utilisant Llvm.

I L'acheminement du projet

J'ai commencé mon TP en suivant les indications données par le sujet et l'ordre dans lequel réaliser les différentes instructions du programme à savoir:

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

J'ai donc réalisé en premier les classes des expressions d'opérations simples comme l'addition, la soustraction, la division, la multiplication ainsi que leur ajout dans le Parser, leur version dans Llvm.java et des premiers tests.
J'ai ensuite voulu réaliser l'instruction d'affectation mais je n'arrivais pas à comprendre utiliser le principe de variable sans savoir d'où venait la table des symboles. J'ai donc réalisé une première version minimaliste de l'ASD partant du début du programme. Voici le squelette que j'ai réalisé et directement codé dans le Parser :

Program -> FUNC*
FUNC -> BLOC+
BLOC -> variable* instructions+
variable -> IntegerVariable
instructions -> affect expression
expression -> (all expression)
factor -> … primary -> …

Ensuite j'ai décidé de coder les classes et leur traduction Llvm de l'affectation et de remonter dans l'arbre en faisant les variables, les déclarations, les blocs, les fonctions (aussi bien en tant qu'expression , instruction ou déclaration de fonction) et le programme pour obtenir une première version fonctionnelle mais quasiment vide de contenue. Je n'ai eu qu'à rajouter les parties manquantes là aussi en remontant dans l'arbre en commençant par les instructions "If then else", "while", "return". Je voulais garder les tableau pour la fin comme l'indiquait l'énoncé mais en réalisant les prototypes, je me suis demandé si un prototype pouvait prendre un tableau comme argument, en cherchant dans les test déjà écrit du TP j'en ai conclu que oui, or je ne voyais pas comment réaliser prototypes sans passer par une déclaration de tableau. J'ai donc pris la décision de réaliser tout ce qui concerne les tableaux dans le Parser et ensuite les coder ; j'ai donc rajouté le fait d'avoir un tableau comme expression, pouvoir affecter une variable dans un tableau, et le fait d'avoir une variable de type tableau. Une fois fini j'ai pu réaliser prototype et modifier Program.java pour pouvoir en contenir. J'ai tenté de réaliser Print et Read mais malgré ma bible depuis le début de ce Tp : "LLVM Language Reference Manual , je n'ai pas réussi à comprendre leur traduction en Llvm.

II Autres problèmes rencontrés

Outre les ceux évoqués ci-dessus, j'ai eu d'autres problèmes que je vais présenter ici. Il ne représente pas la totalité des problèmes que j'ai pu avoir car malheureusement je n'ai pas pensé à tous les noter sur ma feuille de note.

Pour commencer, j'ai eu beaucoup de mal à comprendre le fonctionnement général du projet et surtout la partie traduction Llvm des classes java. J'ai donc "simplement" passer beaucoup de temps à lire la doc de Llvm et faire des aller-retour sur le tp pour comprendre le tout.

Pour me faciliter la traduction Llvm à certains moments comme pour des déclarations de fonction, de tableau, d'apelle de fonction, j'ai réalisé une classe Pair qui représente une variable par son identifiant et son type LLvm. Utilisé en avec une liste, cela me permet d'avoir facilement une liste de variables de plusieurs types d'arguments ou d'index.

J'ai la mauvaise idée de me demander si un tableau pouvait être multidimensionnel et rechercher la réponse directement dans la doc et non dans le sujet de TP, j'ai donc créé une version des tableaux en gérant le fait d'être multidimensionnel. L'incompréhension de réaliser "getElement" prenant plusieurs arguments pour plusieurs index de dimension m'a fait relire le sujet VSL : "*soit de la forme ident[entier]."* réalisant cela, j'en ai conclu que nous ne pouvions avoir un tableau de la forme "*ident[entier][entier][entier]"*, par exemple et que "getElement" n'avait besoin que d'un seul argument.

La traduction Llvm de Prototype reste un mystère car aucune explication n'est fournie dans les sujets, j'ai donc décidé de renvoyer quelque chose de vide mais de l'ajouter dans la table de symbole avec ses arguments.

III Vérification de type

Pour la vérification de type j'ai décidé que les toIR() de chaque classe (sauf Program) devait posséder une table des symboles qui commence au début du programme, qui s'incrémente à chaque déclaration de variable, de fonction ou de prototype et qui descend de plus en plus dans l'AST en tant qu'attribut hérité. Ensuite quand je dois vérifier un type, je regarde si la valeur à vérifier est bien présente dans la table, si oui je compare le type de la valeur et celle trouvée dans la table, si elles ne correspondent pas, je renvoie une erreur TypeException.

Pour permettre la vérification de type, j'ai du modifier la class SymbolTable en y ajoutant plusieurs getter sur ses attributs ainsi qu'une fonction `findArg(String s)` permettant de retrouver une variable dans les paramètres d'une fonction dans la table de symbole.