Compilation
Projet : Compilateur pour le langage SCALPA

Yassine Lambarki El Allioui, Céline Ly, Michael Hofmann, Simon Willer

3 janvier 2021

1 Quadruplet :

On a utilisé un tableau global statique pour stocker le code intermédiaire, qui ensuite va être transformé en MIPS. Le code intermédiaire se présente sous forme de quadruplets (structure quad) à 4 adresses : le type d'instruction, l'opérande 1 et 2 et le résultat. Les opérandes peuvent être soit des constantes (nombres entiers), des chaînes de caractères ou des noms (id) de variables. Les variables temporaires sont traitées comme des variables globales, avec un compteur qui s'incrémente à chaque fois qu'une variable temporaire est créée pour le nom de la variable. Elles sont ajoutées dans la table des symboles. Elles sont principalement créées lorsque l'on fait des opérations arithmétiques, des affections de valeur de retour d'une fonction et pour les constantes numériques comme argument d'une fonction.

L'attribut `next` (vu en cours) est représenté par la structure lpos qui est une liste chaînée.

2 Structures de contrôle :

Les expressions booléennes appelées "cond" dans notre grammaire ont été traitées : les opérations relationnelles, le and, or et not (le xor n'a pas été fait), l'affectation d'une variable à un booléen (ex : a = 1<2). "cond" est du type tf, structure avec un attribut true et false de type lpos. On a séparé les expressions booléennes des expressions arithmétiques car cela évitait des problèmes de typage avec les deux expressions et elles n'ont pas le même type (une de type tf et l'autre lpos). Ainsi dans les structures de contrôle (if then, if else et while), l'utilisateur ne pourra qu'entrer que des booléens comme condition. On a mis une règle de priorité pour le if then et le if else car cela engendrait un conflit décalage/réduction

Les instructions élémentaires return, read, write et les commentaires ont été implémentées

3 Fonction :

L'analyse syntaxique des fonctions gère la déclaration de fonctions successives, l'appel aux fonctions et l'affectation de la valeur de retour d'une fonction. Les arguments sont stockés dans une structure typelist. Lorsqu'une fonction est appelée, on vérifie si elle a été déjà déclarée auparavant et si les arguments correspondent bien aux types des arguments de la fonction déclarée. L'ensemble des structures des

quadruplets prend en compte les fonctions et le code intermédiaire a été fait. Cependant, par manque de temps, seul l'appel des fonctions sans arguments a été traité en MIPS.

4 Typages :

Le typage des variables a été implémenté. Pour chaque affectation, opérations arithmétiques, les actions dans la grammaire vérifient si les 2 opérandes sont bien du même type avec la fonction `chk_symb_type()`. De plus, on vérifie lorsque l'on utilise le token ID que la variable ou la fonction a bien été déclarée auparavant.

5 Table de symboles :

La table des symboles a été implémentée sous la forme d'une table de hachage (fonction basique de hachage par somme de caractère ascii du nom de la variable). Elle est représentée par un tableau de structure (les symboles) qui contiennent un argument représentant un "doublon" d'indice dans la table, utile pour des variables ayant des noms anagramme par exemple. Ces symboles ont 2 types : celui de l'identifieur et le type atomique de la variable.

La déclaration des variables se fait lors de l'analyse syntaxique sous forme d'une chaîne listé contenant le type atomique et le nom de toutes les variables associés (séparé par une virgule).

Les symboles sont majoritairement les variables, qu'elles soient globales ou locales, mais il y a aussi les fonctions et tableaux. Nous n'avons pas géré le système de pile bien que prévu de l'implémenter mais pas pu par manque de temps.

Lors de la création des quad des variables temporaires sont créées (par exemple pour l'affectation d'une addition , b = a + 3, on crée une variable temporaire t0 qui est égale à a+3 et on affecte t0 à b).

Ces variables temporaires sont affectées et prennent donc de la place dans la table de symbole mais cela a été une façon de pouvoir coder rapidement un compilateur fonctionnel. De même concernant les variables locales, elles apparaîtront dans la table des symboles. L'implémentation de table de symbole temporaire qui sont ensuite nettoyées n'existe pas à ce jour.

Des prémices de fonctions d'affectations / opérations arithmétiques entre symboles ont été faites pour d'éventuelles optimisations mais n'ont malheureusement pas été utilisées.

Une gestion de pile a été considérée mais non exploitée dans la table des symboles (argument "portée de la variable " sous forme d'un entier à incrémenter en fonction de la portée dans laquelle se trouve la variable).

6 Option :

Les options implémentées sont -version,-o,-tos qui donne la table des symboles et le code intermédiaire

7 Opérateurs relationnelles et Arithmétiques :

Les opérateurs arithmétiques ont été assez simples à implémenter, à l'aide de quad classiques et d'une union pour la gestion des nombres (sous la forme d'un nom de variable ou directement le nombre). Leur traduction en code MIPS fut une simple traduction des quad élégamment implémentés.

Concernant les Opérateurs Relationnelles c'est à peu près la même chose. Grâce au numéro de quad stocké dans le quadruplet, on peut faire un goto aisément et faire une simple traduction de quad en MIPS.

8 MIPS :

Alors pour le fameux MIPS . Pour gérer le format, d'abord, on affiche des infos concernant le groupe le programme en commentaire. Puis on s'attaque à la section .data où il y'a les variables et les strings. Pour les variables, on loop sur la table de symbole à la recherche de variables ou tableaux puis on les affiche. Les cts strings sont aussi créés si jamais, on détecte un write "string" dans le globalcode. Puis, on

loop sur le globalcode qui contient tous les quadruplets et selon le type de quad , on fais un traduction dans le fichier output. On signale aussi que durant la traduction , on a préféré que le code backend càd en C qui est plus compréhensible et lisible qu'un code MIPS très optimisé. Clairement , des instructions auraient pu être optimisés (comme des addi à la place de ld + add quand un des opérateurs est constant mais ça se discute ..)

9 Tableaux :

Les tableaux sont une partie assez compliquée qui doit être gérée avec prudence. Le défi a surtout été d'implémenter les tableaux sans influer sur ce qui a été fait auparavant. On signale que par souci de simplification, la grammaire a été un peu changée et donc lvalue ne représente qu'élément d'un tableau. Alors d'abord, on déclare un tableau selon le type d'argument int ou uint ou bool. Puis on stocke les dimensions. Il faut signaler que l'affectation des dimensions des tableaux s'est fait dans une structure `dim_list` qui est une liste chainée et qui à chaque fois stocke les `dim_min dim_max` qui sont les intervalles à ne pas dépasser pour chaque dimension. Cette implémentation de liste chainée même si elle est compliquée , elle a été choisi pour gérer tout type de dimensions de tableaux et pas que les 1d ou 2d . Puis le call d'un tableau a aussi été fait avec un soin , grâce à une liste chainée d'indexs . Ces derniers sont vérifiés à chaque fois pour ne pas dépasser les bornes.