Rapport du projet de compilateur SCALPA

Danyl El-Kabir
Jérémy Bach
Nadjib Belaribi
François Grabenstaetter

3 janvier 2021

Table des matières

1 Résumé

L'objectif de ce rapport est de résumer les capacités de notre compilateur SCALPA. Le compilateur génère du code MIPS à partir d'un pseudo code Pascal appellé SCALPA. Nous avons réalisé ce projet de manière incrémentale, en implémentant les différentes fonctions au fur et à mesure.

2 Développement du compilateur

Comme cité précédemment, le développement de ce compilateur s'est fait de manière incrémentale en augmentant petit à petit les capacités du compilateur (en étendant la grammaire).

Dans un premier temps, nous avons réalisé l'analyser lexicale permettant d'analyser un programme SCALPA.

Ensuite, nous avons limité le compilateur aux opérations arithmétiques ainsi que les affectations des variables. Cette version minimale de la grammaire nous à permis de générer notre premier code MIPS et de pouvoir partir sur de bonnes bases.

Une fois cette première étape complètement terminée, nous avons pu élaborer un jeu de tests pour les affectations et les opérations arithmétiques pour enfin ajouter de nouvelles fonctionnalités aux compilateur en étendant davantage la grammaire.

C'est ainsi que nous avons ajouté le support des conditions avec les variables booléennes. L'ajout de ces conditions nous a permis d'ajouter le support des boucles while également.

Par la suite, nous avons étendu la grammaire avec l'ajout du support des tableaux et des fonctions, tout en testant régulièrement nos implémentations.

Après avoir testé le bon fonctionnement du compilateur dans sa plus grande partie, nous avons ajouté un module d'optimisation de code.

3 Spécification complète de notre compilateur SCALPA

Le compilateur est donc capable de générer un programme MIPS R2000 (utilisable sur un simulateur de processeur comme SPIM ou Mars) à partir d'un programme écrit en SCALPA.

Voici une liste non exhaustive des capacités de notre compilateur SCALPA :

Expressions arithmétiques et affectations		
Opérateur	Support	Commentaire
Déclartions de variables		
var a, b : type	X	Déclaration multiple
Affectations et opérations sur les entiers		
:=	X	Affectation entière
+	X	Addition entière
-	X	Soustraction entière
-(unaire)	X	Moins unaire
*	X	Multiplication
/	X	Division
∧	X	Opérateur exponentiel
%	X	Modulo
< ou >	X	Comparaisons entières strictes
<= ou >=	X	Comparaisons entières
=	X	Egalité entières
<>	X	Différence entière
Affectations et opérations sur les booleens		
:=	X	Affectation booléenne
and	X	Opération AND
or	X	Opération OR
xor	X	Opération XOR
not	X	Opération NOT

Structures de controle		
Structure	Support	Commentaire
if expr then instr	X	Conditionnelle simple
if expr then instr else instr	X	Conditionnelle avec else
while expr do instr	X	Conditionnelle while
		expr peut être une condtion sur des booléens ou des entiers

Fonctionnalités des tableaux multidimensionnels		
Instruction	Support	Commentaire
array [-x..y] of type	X	Déclaration avec index et intervales
array [-x1..y1, -x2..y2, -x3..y3] of type	X	Déclaration multidimensionnelle
tab[i,j] :=x	X	Affectation à une case d'un tableau
tab[i,j] :=tab[k,l]	X	Affectation à partir d'un tableau
tab[i,j] < tab[i,j]	X	Comparaisons sur les éléments de tableaux
tab[i,j] or tab[i,j]	X	Opérations booléennes sur les éléments de tableaux

Fonctionnalités des fonctions		
Instruction	Support	Commentaire
function max (a : int, b : int) : int	X	Déclaration avec argument et valeur de retour
function bsort (ref tab : array(1..10) of int, size : int) : unit	X	Passage d'un tableau en argument via référence (effet de bord)
function bsort (tab : array(1..10) of int, size : int) : unit	X	Passage d'un tableau en argument via copie
Fonctions récursives	X	

Commentaires		
Instruction	Support	Commentaire
(* *)	X	Commentaires simples
(* (* *) *)	X	Commentaires imbriqués

Fonctions prédéfinies		
Instruction	Support	Commentaire
read var	X	Lecture d'une entrée dans une variable
write var	X	Affichage de la variable dans la console
write "str"	X	Affichage de constantes dans la console
write tab(i) ou read tab(i)	X	Opérations prédéfinies sur les tableaux

De plus l'analyse lexicale de notre compilateur se fait sans sensibilité à la casse, ce qui laisse une "marge d'erreur" au programmeur, s'il se trompe et écrit While, wHile ou whiLE...etc ce n'est pas grave notre compilateur saura comprendre son code.

Il est également important de noter que les opérations sur entiers respectent l'ordre de priotités usuel sans avoir besoin de forcer l'ordre avec des parenthèses.