

Rapport du projet de compilateur SCALPA

Danyl El-Kabir
Jérémy Bach
Nadjib Belaribi
François Grabenstaetter

3 janvier 2021

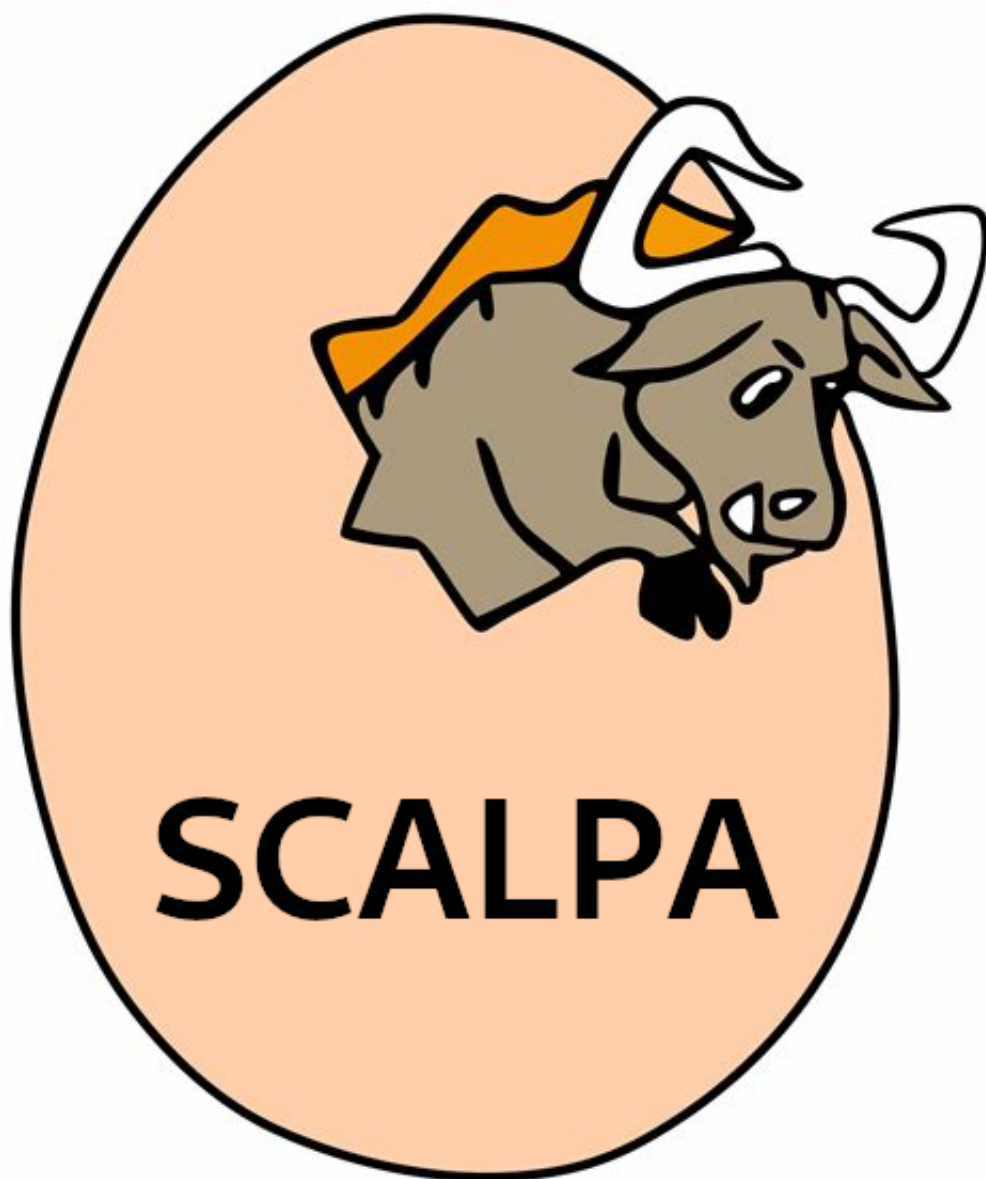


Table des matières

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 appelé 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'analyse 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 a 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 au 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

Affectations et opérations sur les entiers

Opérateur

Support

Commentaire

`:=`

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

4

`:=`

X

Affectation booléenne

`and`

X

Structures de controles
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
Fonctionnalités des tableaux multidimensionnels
Instruction
Support
Commentaire
array(-x1..y1, x2..y2) of type
X
Définition avec index et intervalles
tabe(i,j) :=x
X
Affectation à une case d'un tableau

Le programme écrit en SCALPA peut donc supporter :

- Des affectations d'entiers et booléens sur des variables ou des cases de tableaux.
- Des opérations sur les entiers ou des variables entières (additions, multiplications, soustraction, division, modulo, puissance) tout en respectant les priorités de calcul. (opérations fonctionnelles avec les tableaux comme par exemple $\text{tab}[i] + \text{tab}[j]$ ou encore $\text{tab}(i,j,k) * \text{tab}(l,m,n)$ (tableaux multidimensionnels))
- Des opérations sur les variables booléennes ou des constantes booléennes (OR, AND, XOR, NOT, opérations fonctionnelles avec les tableaux comme par exemple $\text{tab}[i] \text{ xor } \text{tab}[j]$)
- Les opérations de comparaisons sur les entiers sont également implémentés et peuvent être utilisés comme conditions ($<$, $>$, $<=$, $>=$, $=$, $<>$, opérations fonctionnelles avec les tableaux comme par exemple $\text{tab}[i] \text{ xor } \text{tab}[j]$)
- Des opérations sur des fonctions, fonctions récursives avec un passage de paramètres par valeur et par référence.

Ces opérations de comparaisons permettent de créer des conditions de type *if cond then instr else instr* ou *while cond do instr*. Les conditions sont implémentées et permettent de réaliser toutes sortes de boucles. Les conditions peuvent être des comparaisons sur des entiers, ou des opérations sur des booléens par exemple. Nous pouvons également imbriquer les conditions pour vérifier plusieurs assertions par exemple et distinguer des cas. Il est possible de simuler une boucle for avec un while, mais il aurait cependant été possible de l'implémenter directement.

En ce qui concerne les tableaux, ils sont entièrement fonctionnels et permettent de faire des affectations ou des opérations sur des éléments du tableau en fonction du type de variable du tableau. Les tableaux peuvent contenir des booléens ou des entiers et sont déclarés avec des intervalles, ces intervalles correspondent aux index du tableau sur les dimensions.

Les tableaux à multidimensions sont également fonctionnels à condition d'avoir le même type dans toutes les cases. Le contenu d'une case du tableau est interprété comme une variable de ce même type et permet donc de formuler des conditions et des opérations sur un contenu du tableau.

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.

Pour finir, les fonctions peuvent être définies et utilisées dans le programme tout en prenant des arguments en entrée et en retournant des valeurs. Les arguments des fonctions peuvent être des entiers, des booléens ou des tableaux. Les tableaux peuvent être passés en copie ou en référence. Si le tableau est passé en argument par référence, la fonction modifiera le tableau par effet de bord sur le tableau initial. Les fonctions récursives sont également supportées et complètement fonctionnelles.