Langages de programmation, interprétation, compilation
Projet 2025–26

Micro Go

Partie 2 — 23 novembre 2025

La seconde partie du projet consiste en la génération de code MIPS pour les programmes Micro Go.

1 Schéma de compilation

Le schéma de compilation est simple.

Le résultat des expressions sera stocké dans le registre $t0 avec des calculs intermédiaires dans la pile. Concernant l'appel de fonction, les arguments seront passés sur la pile et l'éventuelle valeur de retour sera également placée dans le registre $t0.

On ne demande pas de libérer la place dans le tas pour des structures qui ne seront plus utilisées.

1.1 Points spécifiques

Types primitifs. On supposera que toutes nos constantes entières sont représentables par un entier sur 32 bits signé.

Les booléens sont représentés par des entiers : 0 représente `false` et une valeur non nulle représente `true`.

Une chaîne est représentée par un pointeur vers une chaîne allouée dans le segment de données (car, dans notre fragment, il n'est pas possible de construire de chaînes dynamiquement).

Structures. Une structure est représentée par un bloc mémoire alloué sur le tas. Ce bloc contient les valeurs des champs de la structure, dans un ordre arbitraire. Le compilateur maintient donc une table donnant, pour chaque structure S et chaque champ de S, la position où trouver ce champ dans une valeur de type S.

La valeur `nil`. Elle est représentée par l'entier 0. En particulier, elle est différente de toute adresse d'une structure allouée.

Programme principal `main`. Un programme est une suite de fonctions, il contient une fonction `main` qui sera appelée dans le programme assembleur.

Instruction `Print`**.** L'instruction fmt.Print s'applique à des listes d'expressions de type arbitraire. Il est conseillé d'ajouter au code assembleur engendré des fonctions spécifiques pour les différents types de données puis d'appeler ces fonctions. On pourra dans un premier temps se limiter à l'impression de types simples.

Valeurs multiples. Il y a plusieurs manières de compiler la gestion des affectations et retour de valeurs multiples :

— Transformer les retours multiples en des paramètres supplémentaires de la fonction passés par référence, ce qui correspond en gros aux transformations de programmes ci-dessous. Cela nécessite d'ajouter la gestion des pointeurs et adresses ou bien d'implémenter le passage par référence. On peut obtenir ce comportement par une transformation de programmes Go. Attention dans Micro Go nous n'avons pas considéré la gestion explicite de pointeurs.

```
    func f(...) (tau1,...,taun)          { ... }
=> func f(..., r1 *tau1,...,rn *taun) { ... }

    return e1,...,en
=> *r1 = e1, ..., *rn =en; return

    lv1,...,lvn = f(...)
=> f(..., &lv1,...,&lvn)

    g(f(...))
=> var v1,...,vn; f(..., &v1,...,&vn); g(v1,...,vn)

    return f(...)
=> f(..., r1,...,rn); return
```

— De considérer les n-uplets comme des valeurs allouées sur le tas.

Dans les deux cas, il peut être nécessaire de modifier le type des arbres de syntaxe abstraite pour ajouter les nouvelles constructions. On peut dans un premier temps se limiter à des fonctions qui ne renvoient qu'une valeur.

2 Fichiers complémentaires

— `mips.ml` : contient une interface pour engendrer simplement des programmes MIPS

— `compile.ml` : un squelette de génération de code à compiler

— `typechecker.ml` : a été modifié pour que la fonction de vérification de type renvoie la liste de déclarations présente dans le programme, de manière cohérente avec l'usage fait dans le fichier principal.

Il est possible de faire au moment du typage des transformations de programmes afin de préparer la phase de compilation. On renverra alors le programme transformé.

Ce peut être un nouvel arbre de syntaxe abstraite ou l'ajout de tables stockant les informations glanées à l'analyse de typage.

Vous pouvez modifier cette partie afin de l'adapter à vos besoins.

— `mgoc.ml` a été étendu pour intégrer la phase de génération du code assembleur

Ces 4 fichiers sont fournis dans l'archive : attention à ne pas écraser vos propres versions de `typechecker.ml` et `mgoc.ml` lorsque vous décompressez l'archive.