Master STL - M1 | **Compilation Avancée** | **Université Pierre et Marie Curie**
Module CA (4I504) | *Documents autorisés* | **Année 2016–2017**

Examen du 16 mai 2017

1ère partie (barème indicatif sur 10 points) - à traiter sur une copie séparée

1ère partie : Machine abstraite et langage d'expressions

Le but de ce problème est de réaliser un interprète d'une machine virtuelle à pile en vue de compiler un petit langage d'expressions avec un mécanisme d'appel de fonction.

MACHINE$_{170516}$ est une machine à pile et deux registres : pc (compteur ordinal) et sp (pointeur de pile). Les valeurs manipulées sont soit des entiers, soit des booléens, soit des adresses de code. Les labels permettent de nommer des adresses de code. Pour les branchements, les booléens sont représentés par des entiers : 0 pour `false` et 1 pour `true`.

code	pc	pile	pc	pile
mulint	*adr*	x::y::S	$adr+1$	$x * y$ :: S
divint	*adr*	x ::y::S	$adr+1$	x/y :: S
eq	*adr*	x ::y::S	$adr+1$	$x = y$ :: S
geint	*adr*	x::y::S	$adr+1$	$x >= y$ :: S
acc n	*adr*	a_0 ... ::a_n::S	$adr+1$	a_n:: a_0 ... ::a_n::S
push t v	*adr*	S	$adr+1$	v_t :: S
pop n	*adr*	a_0 ... a_n :: S	$adr+1$	a_n :: S
assign x n	*adr*	a_0 ... a_{n-1} y :: S	$adr+1$	a_0 ... a_{n-1} x :: S
branch L	*adr*	S	L	S
branchif L	*adr*	$true$ :: S	L	S
branchif L	*adr*	x :: S avec $x \neq true$	$adr+1$	S

Question 1 : évaluation de code (1,5 point)

On compile l'expression OCaml suivante : `let my = 142 in if 100 = my then 200 else 3000 ;;`
en :

```
L:      push tint 142
        acc 0
        push tint 100
        geint
        branchif L2
        push tint 3000
        branch L1
L2:     push tint 200
L1:     pop 1
```

Indiquer sur cet exemple l'évolution de la pile et des registres quand on exécute ce code à partir du label L.

Question 2 : schéma de compilation (1,5 point)

Donner le schéma de compilation vers MACHINE$_{170516}$ pour les expressions suivantes du langage :

1. *not expr* où `not` est le NON logique
2. ($expr_1$ && $expr_2$) où `&&` est le ET logique avec court-circuit, il calcule $expr_1$ et ne calcule $expr_2$ que si l'évaluation de $expr_1$ vaut true

On suppose que l'évaluation des instructions du schéma de compilation d'une expression ($expr_i$) calcule un résultat stocké sur le sommet de pile.

Question 3 : évaluateur de MACHINE$_{170516}$ (4 points)

Ecrire un interprète de MACHINE$_{170516}$ en suivant les spécifications données précédemment. Le langage d'implantation est à choisir entre C, Java ou OCaml. Si le calcul ne peut pas se poursuivre (division par zéro, label inconnu, ...) l'évaluateur de MACHINE$_{170516}$ s'arrête. La représentation des valeurs intègre le type dans la valeur, au moins les entiers et les booléens pour le moment.

Vous détaillerez les structures de données et les types utilisés dans votre implantation.

Question 4 : extension de MACHINE$_{170516}$ et du langage (3 points)

On cherche à étendre cette machine pour l'appel de fonction. Pour cela on ajoute un type et deux instructions :

- type `tfun` pour indiquer un type fonctionnel dans un `push`
- `call a` : pour une fonction d'arité a
- `return` : retour de fonction et destruction du cadre courant (ensemble des données nécessaires à l'appel : argument et adresse de retour, ...).

On s'intéresse uniquement aux fonctions globales où toutes les variables sont liées aux paramètres ou sont déclarées localement. Une fonction peut alors être représentée par son label indiquant l'emplacement du code de la fonction. On revient de l'appel par l'instruction `return`, le résultat est sur le sommet de pile (d'avant l'appel). Il est nécessaire de pouvoir gérer les fonctions récursives.

Voici un exemple avec du code en pseudo-OCaml : `let f(x,y) = if x >= y then x else y in f(2,3)` et le code résultant pour la machine MACHINE$_{170516}$:

```
L:   push tint 3
     push tint 2
     push tfun L1
     call 2
     branch L2
L1:  // code de f
     return
L2:  // suite du programme
```

1. Indiquer comment représenter les valeurs fonctionnelles.
2. Indiquer comment représenter le cadre d'appel d'une fonction et comment celui-ci peut retourner à l'instruction suivant l'appel de la fonction.
3. Donner les schémas d'évaluation de ces deux nouvelles instructions `call` et `return`.
4. Implanter-les dans votre interprète de la MACHINE$_{170516}$.
5. **Bonus (2 points)** : Indiquer sans l'implanter mais en étant précis ce qui serait nécessaire d'ajouter à cette machine pour pouvoir traiter des valeurs fonctionnelles générales, fonctions nécessitant un environnement pour gérer les variables libres du corps de la fonction, par exemple des fonctions locales comme dans : `let x = 3 + 4 in let f(z) = z * x in f(10)` Il y a plusieurs solutions envisageables, n'en proposer qu'une seule. Vous pouvez introduire d'autre éléments si nécessaire.