Master STL - M1	Compilation Avancée	Université Pierre et Marie Curie
Module CA (MI190)	*Documents autorisés*	Année 2017–2018

Examen du 29 mai 2018

1ère partie (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 puis de l'étendre avec un mécanisme de rupture de calcul.

MACHINE$_{290518}$ est une machine à pile et trois registres : pc (compteur ordinal), sp (pointeur de pile) et accu (accumulateur). Les valeurs manipulées sont soit des entiers, 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`. La valeur *unit* est représentée par 0.

Instruction	Avant instruction			Après instruction		
	pc	accu	pile	pc	accu	pile
const n	adr	x	S	$adr+1$	n	S
not	adr	x	S	$adr+1$	$not(x)$	S
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
ltint	adr	x	y::S	$adr+1$	$x<y$	S
acc n	adr	x	a_0 ... ::a_n::S	$adr+1$	a_n	a_0 ... ::a_n::S
push	adr	x	S	$adr+1$	x	x :: S
pushacc n	adr	x	a_0 ... a_n :: S	$adr+1$	a_n	x :: a_0 ... a_n :: S
pop n	adr	x	a_0 ... a_n :: S	$adr+1$	x	a_n :: S
assign n	adr	x	a_0 ... a_{n-1} y :: S	$adr+1$	$unit$	a_0 ... a_{n-1} x :: S
branch L	adr	y	S	L	y	S
branchif L	adr	$x=0$	S	$adr+1$	0	S
branchif L	adr	$x\neq 0$	S	L	x	S

Question 1 : évaluation de code

On compile l'expression OCaml suivante : `let x = 2018 in if 50 = x then 10 else 100;;`
en :

```
L:      const 2018
        push
        acc 0
        push
        const 50
        eqint
        branchif L2
        const 100
        branch L1
L2:     const 10
L1:     pop 1
```

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

Question 2 : évaluateur de MACHINE$_{290518}$

Ecrire un interprète de MACHINE$_{290518}$ 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$_{290518}$ s'arrête.

Vous détaillerez les structures de données et les types utilisés dans votre implantation. Vous implémenterez au moins les instructions suivantes : `const n`, `not`, `divint`, `eq`, `acc n`, `push`, `pop n` et `branchif L`. Un bonus sera compté si vous définissez toutes les opérations.

Question 3 : schéma de compilation

Donner le schéma de compilation vers MACHINE$_{290518}$ pour les expressions suivantes du langage :

1.
$$expr_1 \ opbin \ expr_2$$
   où *opbin* est un opérateur binaire (comme $=$, $<$, $+$, $*$).
2.
$$if \ expr_1 \ then \ expr_2 \ else \ expr_3$$

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

Question 4 : extension de MACHINE$_{290518}$ et du langage

On cherche à étendre cette machine avec un mécanisme de rupture de calcul dans les cas où celui-ci ne peut pas se poursuivre. Pour cela on ajoute une pile des récupérateurs, appelée `trapSP`, et trois instructions :

- `raise` déclenchant la rupture de calcul, l'adresse du code à exécuter est située au sommet de la pile des récupérateurs `trapSP`, suivi du pointeur de pile à restaurer;
- `pushtrap` stockant une adresse de code à exécuter, en cas de rupture de calcul, dans la pile des récupérateurs `trapSP`, suivi de l'adresse de pile courante;
- `poptrap` qui dépile une adresse de code et un pointeur de pile de la pile des récupérateurs `trapSP`.

Un récupérateur d'exception correspond alors à deux valeurs : l'adresse du code et le pointeur de pile de la reprise.

1. Donnez les règles d'évaluation de ces 3 instructions, à la manière du tableau indiqué à la première page.
2. Ajoutez à votre évaluateur de MACHINE$_{290518}$ ces 3 instructions.
3. On ajoute au langage la construction `protect`, que l'on utilise de la manière suivante :

   protect expr_1 with expr_2

   et qui s'évalue en la valeur $expr_1$ si le calcul termine normalement et sinon s'évalue en $expr_2$. Donnez alors le schéma de compilation de cette nouvelle expression (*protect expr_1 with expr_2*).
4. En suivant le schéma précédent, produisez le code de l'expression suivante *protect* 320 / 0 *with* 42. On suppose que la division par zéro déclenche l'instruction `raise`.
5. Exécutez le code produit en indiquant les évolutions de la pile et des registres de MACHINE$_{290518}$.