Examen du cours "Machines virtuelles"

Licence 3 – Université Paris Diderot Paris 7 – mai 2012

Durée: 3 heures

Documents autorisés

Le soin apporté à la rédaction et à la présentation ainsi que la rigueur des réponses seront pris en compte dans la notation. Pour faciliter la correction, des commentaires informatifs pourront avantageusement être intégrés à votre code-octet. Le sujet est décomposé en deux parties. La première est une application directe du cours, de durée indicative environ 1h. La seconde est un problème consacré à l'implantation d'une machine à pile et à la vérification de son code-octet.

1 Code-octet OCaml et Java

Exercice 1 *Traduire en code-octet les phrases* OCaml *suivantes :*

1. 2 × 32 / 6 ; ;
2. let x = ref 5 in fun y → !x + 1
3. let z = ref (1,2) in for i = 0 to 3 do z := (snd !z, fst !z) done ; ;

□

Exercice 2 *Soit le code-octet* OCaml *suivant :*

```
        branch L2
L1:
        acc 0
        branchifnot L3
        acc 0
        getfield 1
        push
        acc 0
        branchifnot L4
        acc 0
        push
        offsetclosure 0
        apply 1
        push
        const 1
        addint
        return 2
L4:
        const 1
        return 2
L3:
        const 0
        return 1
L2:
        closurerec 1, 0
```

1. *Exécutez le code-octet suivant dans une machine virtuelle dont l'accumulateur contient la fermeture produit par l'évaluation du code-octet* OCaml *précédent.*

```
const [0: 1 [0: 2 [0: 3 0a]]]
push
acc 1
apply 1
```

2. *Proposez une phrase* OCaml *pour laquelle le code-octet de la question 1 pourrait être le code compilé.*

□

Exercice 3 *Traduire en code-octet les expressions* Java *suivantes sachant que "y" est la variable locale d'indice 0 et "x" est la variable locale d'indice 1.*

1. `y = 31 + 2 * (x / 5);`

2. `y = 1; for (x = 0; x < 10; x++) y *= x;`

□

2 Problème : Implantation d'une machine virtuelle à pile

Dans ce problème, vous devez décrire l'implantation de deux modules d'une machine virtuelle :
- l'interprète de code-octet dont le rôle est d'exécuter le code-octet.
- le vérificateur de code-octet dont le rôle est de vérifier, préalablement à son exécution, qu'un code octet ne peut pas mener à des opérations invalides sur la pile.

Dans la suite, pour décrire ces implantations, vous pourrez utiliser le JAVA, le C ou le CAML.

On introduit une machine virtuelle à pile dont les composantes sont :
- le programme `P` (une suite de code-octets).
- un pointeur de code `PC` valant initialement 0, indice du premier code octet.
- une pile `S` d'entiers 32-bits signés.

Les instructions de la machine virtuelle sont :
- `jump` : dépile un entier x et déplace le pointeur de code en x.
- `jumpif` : dépile un entier x puis un entier y et déplace le pointeur de code en x si et seulement si y est différent de 0.
- `pushi` N : empile l'entier N [au sommet de la pile].
- `pop` : dépile un élément.
- `dup` : duplique l'élément en haut de la pile.
- `swap` : permute les deux éléments en haut de la pile.
- `add`, `mul`, `sub`, `div` : dépile un entier y puis un entier x et empile $x\ \delta\ y$ où $\delta \in \{+, *, -, /\}$.
- `exit` : dépile un entier x et arrête le programme avec x comme code de retour.

Dans la suite, on supposera l'existence d'étiquettes, notées $\ell_1, \ldots, \ell_n$, qui permettent de représenter symboliquement une position dans le code. On définit alors le sucre `pushlbl` ℓ remplacé par `pushi` N où N est la position de l'instruction correspondant à l'étiquette ℓ.

Exercice 4 (Généralités)

1. *Écrivez un programme pour cette machine virtuelle, qui, à partir d'une pile dont le sommet est un entier N, produit une pile dont le sommet vaut $\sum_{k=1}^{N} k^2$.*
2. *Décrivez une façon systématique de programmer dans cette machine virtuelle des calculs de la forme $\sum_{k=1}^{N} e$ où e est une expression arithmétique dépendante de k.*

Dans le cas d'un code octet qui a été téléchargé par exemple, il est nécessaire d'effectuer un certain nombre de vérifications pour s'assurer que le programme s'évalue sans erreur. Dans cette partie, vous allez implémenter un (modeste) vérificateur

```
    pushlbl L1
    jump
    pushi 1
L1:
    sub
    pushlbl L1
    jumpif
    exit
```

```
L1:
    pushi 1
    sub
    dup
    pushlbl L1
    jumpif
    exit
```

```
L0:
    dup
    pushlbl L2
    jumpif
    pop
    pushi 4
L1:
    pushi 4
    sub
    pushlbl L1
    jumpif
    pop
    pushi 42
    exit
L2:
    pushi 1
    sub
    pushlbl L0
    jump
```

```
L0:
    dup
    pushi 1
    sub
    dup
    pushlbl L0
    jumpif
L1:
    pushi 0
    add
    dup
    pushlbl L1
    jumpif
    exit
```

On s'intéresse maintenant aux programmes qui vérifient le critère suivant :

(a) Il est possible d'associer une taille N_ℓ à toute étiquette ℓ de sorte qu'à chaque fois que l'interprète passe par cette étiquette ℓ, la taille de la pile doit être N_ℓ.

3. On considère une pile initiale d'une certaine taille d. Montrer que si un programme vérifie le critère (a), il est alors possible de calculer la taille de la pile en tout point [illegible]