Examen du cours “Machines virtuelles” ($2^{\text{ème}}$ session)
Licence 3 – Université Paris Diderot Paris 7

Durée: 3 heures

Tout document non manuscrit autorisé.

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, à laquelle vous devez consacrer au plus une heure. 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 O'Caml et Java

Exercice 1 *Traduire en code-octet les phrases* O'CAML *suivantes :*

1. `1 + 42 / 6 ; ;`

   Réponse:

   ```
   const 6
   push
   const 42
   divint
   push
   const 1
   addint
   ```

2. `let x = ref 0 in fun y → !x + 1`

   Réponse:

   ```
   const 0
   makeblock 1, 0
   push
   acc 0
   closure L1, 1
   return 2
L1: envacc 1
   getfield 0
   offsetint 1
   return 1
   ```

3. `let z = ref (0, 1) in for i = 0 to 2 do z := (snd !z, fst !z) done ; ;`

   Réponse:

```
        const [0: 0 1]
        push
        const 0
        push
        const 2
        push
        push
        acc 2
        gtint
        branchif L2
L1:
        acc 2
        getfield 0
        push
        acc 3
        getfield 1
        makeblock 2, 0
        assign 2
        acc 1
        push
        offsetint 1
        assign 2
        acc 1
        neqint
        branchif L1
L2:
        const 0a
        return 4
```

□

Exercice 2 *Soit le code-octet* O'CAML *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* O'CAML *précédent.*

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

Réponse:

2. *Proposez une phrase* O'CAML *pour laquelle le code-octet de la question 1 pourrait être le code compilé.*
Réponse:

```
let rec length l = match l with [] → 0 | [_] → 1 | _ :: xs → 1 + length xs ; ;
```

□

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 = 21 + 2 * (x / 10) ;`

Réponse:

```
0: iconst_0
1: istore_0
2: iconst_1
3: istore_1
4: bipush 21
6: iconst_2
7: iload_1
8: bipush 10
10: idiv
11: imul
12: iadd
13: istore_0
14: return
```

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

Réponse:

```
0: iconst_0
1: istore_0
2: iconst_1
3: istore_1
4: iconst_1
5: istore_0
6: iconst_0
7: istore_1
8: iload_1
9: bipush 10
11: if_icmpge 30
14: iload_0
15: ifne 22
18: iconst_1
19: goto 23
22: iconst_0
23: istore_0
24: iinc 1, 1
27: goto 8
30: return
```

□

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 code `C`, une liste de code-octets.
- un pointeur de code `PC` valant initialement 0, indice du premier code octet.
- une pile d'entiers de 32 bits signés `S`.

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 : pousse l'entier $N \in [0..2^{28}[$ 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.*

□

Exercice 5 (Interprète de code-octet)

1. *Pour chaque composante de la machine virtuelle décrite plus haut, spécifiez le type de données qui pourrait la représenter dans une implantation ainsi que les opérations que vous supposez données sur ces types de données.*
2. *Rappelez les différentes étapes de l'interprétation d'un programme écrit en code-octets.*
3. *En utilisant les opérations décrites dans la réponse à la question 1, écrivez une fonction d'interprétation pour cette machine virtuelle. Attention à factoriser votre code !*

□

Exercice 6 (Vérification de code-octet) *Avant d'exécuter du code-octet provenant d'une source inconnue (c'est le cas d'un code-octet qui a été téléchargé par exemple), la JVM fait un certain nombre de vérifications pour s'assurer que le programme s'évalue sans erreur. Dans cette partie, vous allez implémenter un (modeste) vérificateur de code-octet pour la machine virtuelle à pile décrite plus haut en vous focalisant sur l'utilisation correcte de la pile.*

1. *Quelles opérations de la machine virtuelle peuvent échouer ? Pour chacune d'elles, exprimez les conditions sur la forme de la pile qui sont nécessaires à son bon fonctionnement.*
2. *Parmi les 4 programmes suivants, lesquels font un mauvais usage de la pile ? Pourquoi ?*

```
    pushi 40
    pushlbl L1
    jump
    pushi 1
L1:
    sub
    pushlbl L1
    jumpif
    exit
```

```
    pushi 40
L1:
    pushi 1
    sub
    dup
    pushlbl L1
    jumpif
    exit
```

```
    pushi 40
L0:
    dup
    pushlbl L1
    jumpif
    pop
    pushi 4
L00:
    pushi 4
    sub
    pushlbl L00
    jumpif
    pop
    pushi 42
    exit
L1:
    pushi 1
    sub
    pushlbl L0
    jump
```

```
    pushi 10
L0:
    dup
    pushi 1
    sub
    jumpif l0
L1:
    pushi 0
    add
    jumpif L1
    exit
```

On définit les critères de bonne formation suivants :

- *On peut associer une taille N_ℓ à toute étiquette ℓ : à chaque fois que l'interprète passe par cette étiquette, on sait que la taille de la pile est N.*
- *Pour chaque instruction, la taille de la pile vérifie les conditions nécessaires de bon fonctionnement des opérations.*

3. *Expliquez pourquoi si un programme P vérifie ces critères alors on peut calculer la taille de la pile en tout point du programme à une constante C près.*
4. *Proposez un algorithme pour vérifier que ces critères sont vérifiés.*
5. *Parmi les 4 programmes précédents, quels sont ceux qui sont acceptés par votre algorithme ?*
6. *Est-ce que tous les programmes rejetés par votre algorithme provoquent réellement une erreur de manipulation de pile lors de leur exécution ?*
7. *Proposez une amélioration de cette méthode de vérification.*

□