Université Paris Cité
L3 Informatique

Machines Virtuelles
Année 2023-2024

TP n°5

Ocamlrun, fonctions

Le but de ce TP est de manipuler des instructions de la machine virtuelle Ocamlrun. N'hésitez pas à consulter la description du jeu d'instructions disponible à :

cadmium.x9c.fr/distrib/caml-instructions.pdf

Un resumé des instructions utiles pour ce TP est donné ci-dessous :

`envacc n` écrit sur `accu` la valeur du n-éme élement de la fermeture courante (c.à-d. pointé par `env`) ;

`apply n` pour $n \leq 3$ suppose que `accu` contient un pointeur vers une fermeture, écrit après n éléments de la pile `S` les valeurs de `pc`, `env` et `extra_args`, le registre `env` reçoit `accu`, le registre `extra_args` reçoit $n-1$, le point d'exécution `pc` saute à l'adresse de l'instruction dans la fermeture ;

`appterm n, s` (appel terminal d'une fonction) suppose que `accu` contient un pointeur vers une fermeture, garde n éléments sur la pile `S` et efface les autres $s-n$ éléments. Le registre `env` reçoit `accu`, le registre `extra_args` reçoit $n-1$, le point d'exécution `pc` saute à l'adresse de l'instruction dans la fermeture ;

`return n` on enlève n éléments obsolètes de la pile. Deux cas viennent ensuite :

- si `extra_args` vaut 0 : c'est le cas usuel d'une fonction ayant reçu exactement assez d'arguments. On termine la fonction en enlevant de la pile `S` trois éléments de plus, qui servent à restaurer respectivement les registres `pc` (augmenté de 1)/`env`/`extra_args` ;
- si `extra_args`> 0, cela signifie que la fonction courante a retourné dans `accu` une fermeture, qu'il faut maintenant exécuter sur les arguments restants : on décroit `extra_args` de 1, `env` reçoit `accu` et on saute au code contenu dans cette fermeture (voir `apply`) ;

`closure Lk n` crée une fermeture (bloc de tag 247) contenant le label `Lk` suivi de n éléments d'environnement ôtés de `accu+S` ;

`grab n` teste si une fonction n-aires a eu assez d'arguments, deux cas :

- si `extra_args`$\geq n$, on continue avec `extra_args` décrémenté de n
- sinon on place les (`extra_args`$+1$) arguments présents sur la pile dans une fermeture de la forme :

Header	Lbl de `RESTART`	pointeur env	arg0	...	argk

`restart env` doit pointer vers une fermeture de la forme précédente. On replace alors les arguments sur la pile, et on incrémente `extra_args` d'autant et on récupère l'environment de la fermeture ;

`closurerec o, n` comme `closure` + `push` ;

`offsetclosure n` place dans `accu` le pointeur contenu dans le registre `env`, décalé de n cases.

Exercice 1. Deviner les expressions OCAML qui génèrent le code suivant :

(i).

```
    closure L1, 0
    push
    const 3
    push
    acc 1
    apply 1
    offsetint 1
    return 2
L1: acc 0
    push
    acc 1
    mulint
    offsetint 1
    return 1
```

(ii).

```
    const 2
    push
    closure L1, 0
    push
    const 3
    push
    acc 1
    apply 1
    pop 1
    mulint
    return 1
L1: acc 0
    push
    const 2
    mulint
    return 1
```

(iii).

```
    closure L2, 0
    push
    closure L1, 0
    push
    acc 1
    push
    acc 1
    appterm 1, 4
L1: const 3
    push
    acc 1
    apply 1
    offsetint 1
    return 1
L2: acc 0
    push
    const 2
    mulint
    return 1
```

Exercice 2. Pour chacune des listes d'instructions suivantes, calculer à la main l'évolution de la configuration de ocamlrun (`pc`, `accu`, `env`, `extra_args`, `S`, `tas`) et deviner l'expression de ocaml qui l'a générée :

(i).

```
    closure L1, 0
    push
    const 1
    push
    acc 1
    apply 1
    push
    const 3
    push
    acc 2
    apply 1
    addint
    return 2
L1: acc 0
    offsetint 2
    return 1
```

(ii).

```
    const 1
    push
    acc 0
    closure L2, 1
    push
    closure L1, 0
    push
    acc 1
    appterm 1, 4
L1: acc 0
    offsetint 1
    return 1
L2: envacc 1
    push
    acc 1
    apply 1
    push
    envacc 1
    addint
    return 1
```

Exercice 3 (Fonctions récursives)**.** Deviner les expressions OCAML qui génèrent le code suivant :

(i).

```
    closurerec 1, 0
    const 3
    push
    acc 1
    apply 1
    offsetint 1
    return 2
L1: acc 0
    push
    offsetclosure 0
    apply 1
    push
    acc 1
    addint
    return 1
```

(ii).

```
    closurerec 1, 0
    const 3
    push
    acc 1
    apply 1
    offsetint 1
    return 2
L1: acc 0
    push
    const 0
    eqint
    branchifnot L2
    const 0
    return 1
L2: const 2
    push
    acc 1
    offsetint -1
    push
    offsetclosure 0
    apply 1
    mulint
    return 1
```

(iii).

```
    const 1
    push
    acc 0
    closurerec 1 2, 1
    const 3
    push
    acc 2
    apply 1
    offsetint 2
    return 4
L1: acc 0
    push
    offsetclosure 2
    apply 1
    offsetint 1
    return 1
L2: envacc 1
    push
    acc 1
    push
    offsetclosure -2
    apply 1
    addint
    return 1
```

Exercice 4 (Fonctions n-aires)**.** Calculer à la main l'évolution de la configuration de ocamlrun sur le code-octet générée par l'expression suivante :

— `let f x y z = x * y + z in let a = f 1 in a 2 3`

Exercice 5 (Fonctions n-aires)**.** Comparer le code-octet généré par `ocaml -dinstr` sur les deux expressions suivantes :

— `let sum x = (let g y = x+y in g) in sum 1 2`

— `let sum x y = x+y in sum 1 2`

Quelle est la différence ? Calculer à la main l'évolution de la configuration de ocamlrun dans le deux cas (utiliser la description du jeu d'instructions pour trouver le comportement de `restart`, `grab` et `apply`).

Exercice 6 (Récursion terminale)**.** Comparer le code-octet généré par `ocaml -dinstr` sur les deux expressions suivantes :

— `let rec f x = if x = 0 then 0 else x+(f (x-1)) in f 2`

— `let rec f_term (x,a) = if x = 0 then a else f_term (x-1, x+a) in f_term (2,0)`

Quelle est la différence ? Calculer à la main l'évolution de la configuration de ocamlrun dans le deux cas (utiliser la description du jeu d'instructions pour trouver le comportement de `appterm`, `getfield`, `offsetclosure`, etc...).