
Introduction aux machines virtuelles

Cours 4 : les fonctions dans la Machine Virtuelle Caml

Pierre Letouzey

`Prenom.Nom@pps.jussieu.fr`

Dernières instructions à voir

CLOSURE

APPLY

RETURN

PUSH_RETADDR

ENVACC

CLOSUREREC

OFFSETCLOSURE

GRAB

RESTART

APPTERM

CamL: points-clés concernant les fonctions

- fonctions anonymes : $(\text{fun } x \rightarrow 1+x)$
- fonctions en arguments et/ou en résultat:

```
let funincr f = (fun x → 1 + f x)
```
- accès hors des frontières de la fonction :

```
let a = 10 in (fun x → a+x)
```
- récursivité (éventuellement mutuelle)
- application partielle : $\text{let add10} = (+) 10$
- une optimisation: les appels terminaux

Caml: points-clés concernant les fonctions

- fonctions anonymes :
 - ⇒ une zone de code (comme les autres fonctions)
- fonctions en arguments et/ou en résultat:
 - ⇒ des pointeurs vers des blocs
- accès hors des frontières de la fonction :
 - ⇒ des environnements
- récursivité (éventuellement mutuelle): cf plus tard
- application partielle : ⇒ hibernation (grab/restart)
- une optimisation: les appels terminaux : cf plus tard

Un exemple élémentaire de fonction

```
%> ocaml -dinstr
```

```
# let f x = 1+x in (f 4)*2;;
```

```
closure L1, 0
```

```
push
```

```
const 2
```

```
push
```

```
const 4
```

```
push
```

```
acc 2
```

```
apply 1
```

```
mulint
```

```
return 2
```

```
L1: acc 0
```

```
push
```

```
const 1
```

```
addint
```

```
return 1
```

Description informelle

- L1 est un **label**, ou position dans le code
- Le vrai bytecode contient plutôt des décalages (ou offsets)
- code avant L1: code d'une sorte de fonction "main"
- code à partir de L1: code de f
- **CLOSURE L1, 0** : fabrique un bloc contenant le label L1 et retourne un pointeur ptr vers ce bloc
- **APPLY 1** : lance l'exécution de f (car accu contient ptr) avec 1 argument dans la pile (valant 4).
- **ACC 0** : f accède à son premier argument
- **RETURN 1** : f rend la main, en nettoyant 1 case de pile. Le résultat 5 est contenu dans accu.

Sauvegardes/restaurations lors des appels/retours

Magie noire: où repartir à la fin d'un appel de fonction ?

⇒ Besoin de mémoriser pc lors de l'appel.

De plus, lors d'appels imbriqués, une case ne suffit pas.

⇒ Le pc de retour est sauvé dans la pile, juste après les arguments. **APPLY1..APPLY3** le fait automatiquement, mais pas **APPLY n**, qui doit être précédé d'un **PUSH_RETADDR**.

En fait, deux autres éléments sauvegardés en plus: env et extra_args, cf. plus tard.

RETURN n enlève n éléments de la pile, puis espère trouver ensuite un triplet pc/env/extra_args à restaurer (+ saut à pc).

Environnements

Une fonction peut utiliser des objets disparus lors de l'appel:

```
let f =  
  let c = 10 in  
  let g x = x*x in  
  fun x → c + g x  
in 1 + f 1
```

Le bloc (ou clôture) de `f` sera fait ici avec **CLOSURE L1, 2**

Il contient un environnement (de taille 2) après le label de `f`:

Header: size 3, tag 247	Lbl de f	Val de c	Ptr vers g
-------------------------	----------	----------	------------

Le registre `env` pointe vers la clôture de la fonction courante.

Accès à `c`: **ENVACC 1** (suit le registre `env`, case 1)

Accès à `g`: **ENVACC 2**

Récurtivité

Dans le cas d'une fonction récursive f non-mutuelle, un appel récursif se fait:

- en chargeant dans `accu` un pointeur sur la clôture de f :
`OFFSETCLOSURE0`
- en faisant un `APPLY` comme précédemment.

Pour les fonctions récursives mutuelles, Caml utilise une même clôture pour tout un paquet mutuel (cf dessin au tableau).

`OFFSETCLOSURE n` permet ensuite de retrouver un pointeur vers une fonction récursive "voisine".

Applications partielles

Si une fonction attend deux arguments et n'en reçoit qu'un, l'exécution ne peut avoir lieu. En attendant l'argument manquant, on doit geler l'exécution, via une clôture.

Le registre `extra_args` vaut en permanence `nb_args - 1`

Au lancement, toute fonction reçoit au moins 1 arg.

GRAB `n` mis au début d'une fonction vérifie si les `n` args suivants sont présents. OK : exécution normale. KO : retourne immédiatement une clôture avec les args présents:

Header	Lbl de RESTART	ptr env	arg ₀	...	arg _k
--------	----------------	---------	------------------	-----	------------------

Un **RESTART** précède toujours le **GRAB**. Le jour où la clôture est appliqué, il s'exécute, et restaure les args sauvés.

Appels terminaux

Fréquemment, un appel via **APPLY** est juste avant un **RETURN**.

Gâchis en pile et en temps:

- le **APPLY** sauvegarde dans la pile `pc/env/extra_args` pour permettre le retour à l'instruction suivante
- cette instruction suivante, le **RETURN**, restaure une autre zone `pc/env/extra_args` plus ancienne et saute à ce `pc`

L'idée est alors de grouper **APPLY** n + **RETURN** m en une instruction **APPTERM** $n, n+m$ qui évite la sauvegarde au milieu.

Dans le cas de fonctions récursives terminales (tail-recursive), le calcul se fait en pile constante... (DEMO).

Description précise des instructions

- **CLOSURE** o, n : crée une clôture (bloc de tag 247) contenant le label correspondant à l'offset o suivi de n éléments d'environnement ôtés de **accu+pile**.
Un pointeur vers cette clôture est placé dans **accu**.
- **CLOSUREREC** $o_1 \dots o_k, n$: si un seul offset o_1 , cf. **CLOSURE+PUSH**. Sinon, fabrique la clôture suivante, en prenant n éléments d'environnement $e_1 \dots e_n$ dans **accu+pile**.

hdr	pc+ o_1	hdr	pc+ o_2	...	hdr	pc+ o_k	e_1	...	e_n
-----	-----------	-----	-----------	-----	-----	-----------	-------	-----	-------

accu reçoit un pointeur vers cette clôture, et on fait **PUSH**

- **PUSH_RETADDR** : empile successivement les valeurs des registres `extra_args`, `env` et `pc`

Description précise des instructions

- **APPLY** *n* : **accu** doit contenir un pointeur vers une clôture, et la pile doit contenir *n* arguments suivi d'une zone où sont sauvegardés les **pc/env/extra_args** courant.
Le registre **env** reçoit **accu**.
Le registre **extra_args** est positionné à *n*-1.
Le point d'exécution **pc** saute au label dans la clôture.
- **APPLY1** ... **APPLY3** : cf **APPLY**, sauf que ces variantes sauvent elles-mêmes **pc/env/extra_args** derrière les arguments: pas besoin de **PUSH_RETADDR** avant.

Description précise des instructions

- **RETURN** n : De toute façon, on commence par enlever n éléments obsolètes de la pile. Deux cas 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 trois éléments de plus, qui servent à restaurer respectivement les registres `pc/env/extra_args` (en particulier saut à `pc`).
 - Si **extra_args** > 0 , cela signifie que la fonction courante a retourné dans **accu** une clôture, qu'il faut maintenant exécuter sur les arguments restants: on décroît **extra_args** de 1, **env** reçoit **accu** et on saute au code contenu dans cette clôture (cf. **APPLY**).

Description précise des instructions

- **ENVACC n** : accède au **n**-ième champs de la clôture pointé par le registre **env**.
- **ENVACC1...ENVACC4** : idem pour **n=1..4**
- **OFFSETCLOSURE n** : place dans **accu** le pointeur contenu dans le registre **env**, décalé de **n** cases.
- **OFFSETCLOSURE0, OFFSETCLOSURE2, OFFSETCLOSUREM2**: cf précédent avec **n=0,2,-2**
- et toutes les instructions de cette page ont des variantes **PUSHXYZ**

Description précise des instructions

- **APPTERM** n,m : moralement **APPLY** n + **RETURN** $(m-n)$.

Plus précisément, la pile doit avoir la forme:

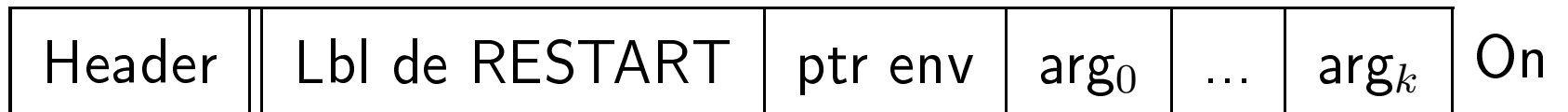
- n arguments pour l'appel à effectuer
- $(m-n)$ cases inutiles
- peut-être de vieux arguments (en nombre **extra_args**)
- une zone de sauvegarde `pc/env/extra_args` pour sortir de la fonction courante

Les cases inutiles sont supprimées, et on décale les arguments. **extra_args** est incrémenté de $n-1$. Comme pour **APPLY**, **env** reçoit **accu** et on saute au label dans la clôture.

- **APPTERM1** m ... **APPTERM3** m : cf précédent.

Description précise des instructions

- **GRAB n** : 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 clôture de la forme:



retourne alors cette clôture dans **accu** et on restaure pc/env/extra_args (cf. **RETURN**).

- **RESTART** : **env** doit pointer vers une clôture de la forme précédente. On replace alors les arguments sur la pile, et on incrémente **extra_args** d'autant.

Machine Virtuelle Caml, Conclusion

Quelques portions “subsidiaries” de la C.V.M. non traitées (env global, exceptions, objets).

Important: représentation des données en mémoire (par blocs).

Langage fonctionnel \Rightarrow soin apporté au traitement des fonctions

Questions ?

Une référence ancienne, plus tout à fait d’actualité, mais encore très instructive:

The ZINC experimentation, Xavier Leroy

<http://caml.inria.fr/pub/papers/xleroy-zinc.ps.gz>