Introduction à la Compilation
Cours 12 : Compilation des fonctions de premiŁre classe

Yann RØgis-Gianas
yrg@pps.univ-paris-diderot.fr

PPS - UniversitØ Denis Diderot Paris 7

Une machine abstraite pour les langages fonctionnels

Retour àune machine àdeux piles

- Deux mØcanismes sont essentiels pour traiter les fermetures :
  - pour les construire : il fautcapturer (un fragment de) l'environnement lexical ;
  - pour les appliquer : il fautrestaurer cet environnement lexical.
- L'Øvolution de la pile des rØsultats intermØdiaires est maintenue indépendamment de celle de l'environnement lexical.

) On s'intØresse donc de nouveau une machine à deux piles $_v$ et $_r$.

Instructions liØes aux fermetures

- close :
  DØpile de $_v$ une pile de variables et un pointeur de code et empile une fermeture qui reprØsente une paire de ces deux composantes.
- open :
  DØpile une fermeture composØe d'un pointeur de code et d'une pile ^0_v, pousse la pile ^0_v et pc au sommet de $_r$.

Compilation du -calcul vers la machine virtuelle

- $C(\ ; x) = \text{getvar}\ \ \text{pos}(x; \ )$

- $C(\ ; \text{fun}\ x\ )\ \ e) =$

  $\quad\quad\ \ \text{branch}\ \ \text{‘}^{0}$

  $\text{‘}\ :\ \ \text{define}$

  $\quad\quad\ \ C(\ \ x; e)$

  $\quad\quad\ \ \text{undefine}$

  $\quad\quad\ \ \text{swap}$

  $\quad\quad\ \ \text{restore}$

  $\quad\quad\ \ \text{swap}$

  $\quad\quad\ \ \text{ubranch}$

  $\text{‘}^{0}:\ \ \text{remember}\ \text{‘}$

  $\quad\quad\ \ \text{capture}\ \ \text{pos}(FV(e) \cap \{x\}; \ )$

  $\quad\quad\ \ \text{close}$

- $C(\ ; c) = \ldots$

- $C(\ ; e_1\ e_2) =$

  $\quad\quad\ \ \text{remember}\ \text{‘}$

  $\quad\quad\ \ \text{capture}\ \ \ \ v$

  $\quad\quad\ \ C(\ ; e_2)$

  $\quad\quad\ \ C(\ ; e_1)$

  $\quad\quad\ \ \text{open}$

  $\quad\quad\ \ \text{restore}$

  $\quad\quad\ \ \text{ubranch}$

  $\text{‘}\ :\ \ \ldots$

Exemple

$C((\text{fun } x) \; x) \; 2) =$

```
        remember ‘00
        capture v
        remember 2
        branch ‘0
‘ :     define
        getvar 0
        undefine
        swap
        restore
        swap
        ubranch
‘0 :    remember ‘
        capture ;
        close
        open
        restore
        ubranch
‘00 :   ...
```

Exemple

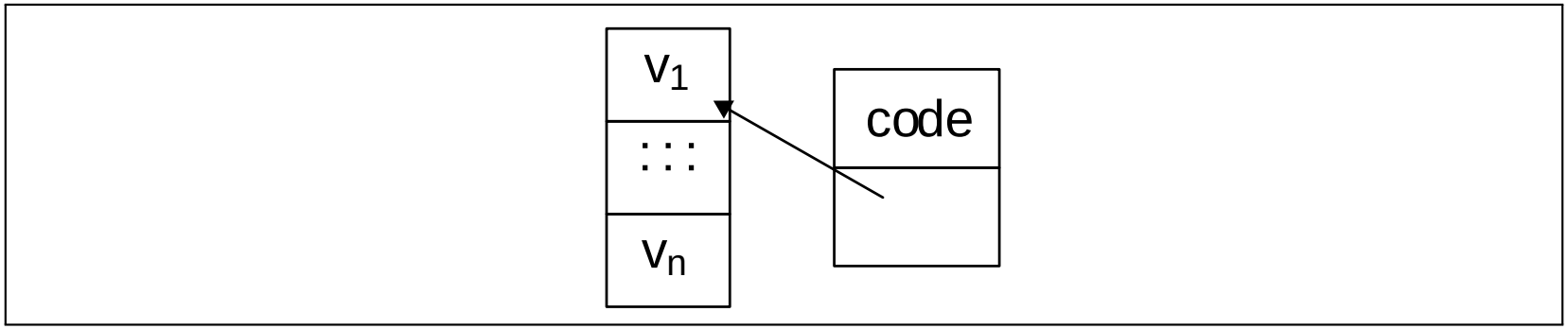
$C((\text{fun } x \; y) \; 2) =$

```
        remember ‘00
        capture v
        remember 2
        branch ‘0
‘:      define
        getvar 0
        undefine
        swap
        restore
        swap
        ubranch
‘0:     remember ‘
        capture y
        close
        open
        restore
        ubranch
‘00:    ...
```

ReprØsentation d'une fermeture

- Du point de vue de la machine virtuelle, une fermeture est un couple form
  d'un pointeur de code et d'une pile de variables capturØes.
- On peut choisir sa reprØsentation interne : il su t que les instructions
  capture ; close ; open et restore se comportent de façon cohØrente.

Deux mots : le PC et un pointeur



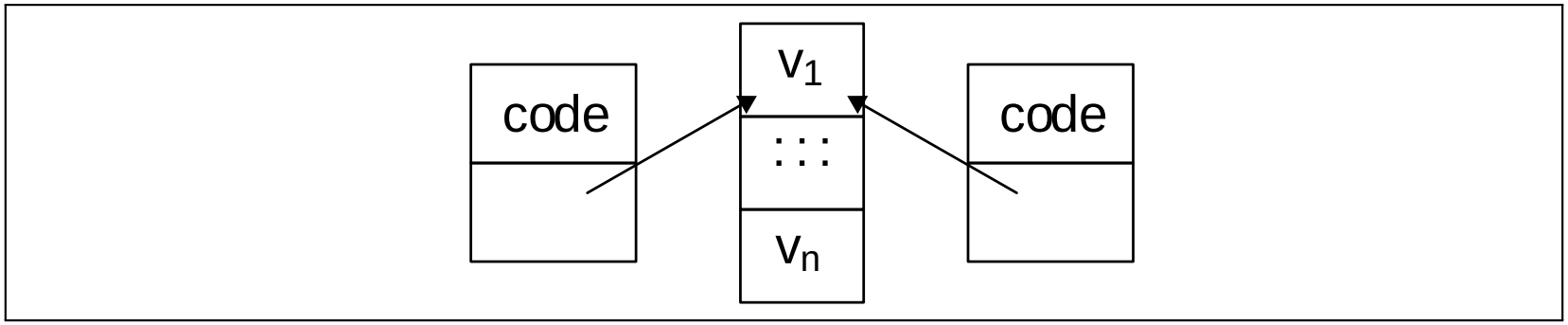
- taille(fermeture) = 2 * taille(pointeur) + taille(environnement)
- capture : pousse un pointeur sur v au sommet de r.
- restore : ØcrasE le pointeur courant de v par celui situØ au sommet de r.
- open : copie un pointeur de code et un pointeur au sommet de la pile p
- close : copie un pointeur de code et un pointeur du sommet de la pile p dans une paire placØe au sommet de p.

X Gain en espace mØmoire.

Impose une structure de pile persistante dont on puisse prendre l'adresse.

AccŁs lent aux variables (il faut suivre des pointeurs allouØs dynamiqueme

Deux mots : le PC et un pointeur



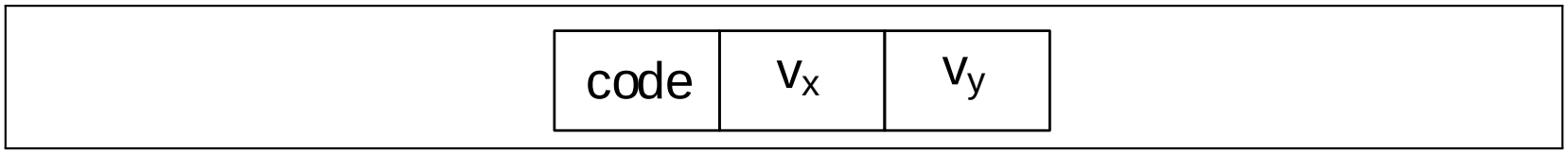
- taille(fermeture) = 2 * taille(pointeur) + taille(environnement)
- capture : pousse un pointeur sur v au sommet de r.
- restore : Øcrase le pointeur courant de v par celui situØ au sommet de r.
- open : copie un pointeur de code et un pointeur au sommet de la pile p.
- close : copie un pointeur de code et un pointeur du sommet de la pile p dans une paire placØe au sommet de p.

X Gain en espace mØmoire.

Impose une structure de pile persistante dont on puisse prendre l'adresse.

AccŁs lent aux variables (il faut suivre des pointeurs allouØs dynamiqueme

Un unique bloc



- taille(fermeture) = taille(pointeur) + taille(environnement)
- capture : alloue un tableau au sommet de $_f$ pour y recopier $_v$.
- restore : fait pointer $_v$ vers le tableau au sommet de $_f$.
- open : copie un pointeur de code et un pointeur de tableau au sommet $_r$ de
- close : copie un pointeur de code et un pointeur vers un tableau de la pile $_r$ vers une paire.

ReprØsentation coßteuse en mØmoire.

X AccŁs aux variables plus e cace.

Les fonctions rØcursives

- Comment compiler l'expression suivante ?

$$\text{fix}\ (\text{fun}\ \text{loop}^0)\ \ \text{fun}\ x\ )\ \ \text{loop}^0\,(x + 1))$$

- Quelles formes prennent les fermetures produites par l'opØrateur fix ?

Comment reprØsenter les fermetures rØcursives ?

- Pour compiler une valeur rØcursive, on commence par allouer une fermetu
dans laquelle les appels rØcursifs sont reprØsentØs par les occurrences d
variables libres.
- Les appels rØcursifs semblent donc nØcessiter un accŁs àl'environnemen
- Pour construire une fermeture rØcursive utilisØe par cette mØthode,
on procŁde en deux temps :
  1. On alloue une fermeture avec une rØfØrence par dØfaut associØe à
  l'environnement.
  2. On met àjour la rØfØrence à dans l'environnement avec la vØritable
  rØfØrence de la fermeture.
- On crØØ un cycle.

Comment reprØsenter les fermetures rØcursives ?

- La reprØsentation àl'aide d'un cycle introduit une indirection :
  - peu naturelle (un appel rØcursif est dØjàdans la bonne fermeture) ;
  - sous-optimal (le saut àune adresse inconnue).
- La fermeture courante est identique àcelle rØcupØrØe dans l'environneme
- On compile les appels rØcursifs d'une façon spØci que pour rØutiliser la fermeture courante.

Et les fonctions mutuellement rØcursives ?

```
...
let rec g z = h z in
and h t = g t in
  ...
```

- Moyennant l'ajout (sans di cultØ) de n-uplets ànotre langage, on peut Øcrire ce programme :

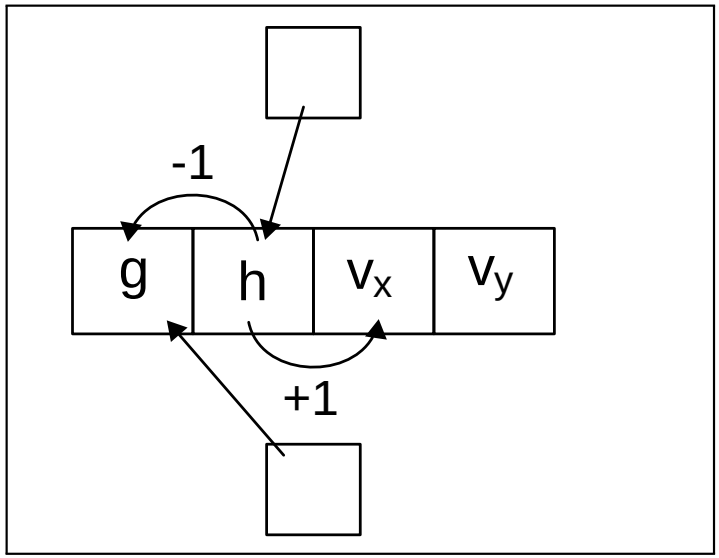
```
let m = fix (fun m) (fun z) (snd m) z; fun t) (fst m) t))
let g = fst m
let h = snd m
  :::
```

- On construit alors N fermetures mutuellement rØcursives, une pour chaque fonction.

) En fait, on peut faire beaucoup mieux !

Compilation e cace des fonctions mutuellement récursiv

```
let f x y =
   let rec g z = z - h x
   and h t = g y + t
   in
    ...
```



- Il est possible de partager un unique bloc pour représenter toutes les fonctions mutuellement récursives :
  1. l'environnement est formé à partir de l'union des variables libres des fonctions
  2. les pointeurs de code des di érentes fonctions sont agglomérés en tête du bloc.
- Chaque fonction se voit a ecter un pointeur qui peut être placé après la tê de la fermeture.
- Pour plus de détails :
  Compiling with continuations, Cambridge University Press, pages 107-108.