Master STL - M1 **Caompilation Anvancée** **Université Pierre et Marie Curie**
Module CA (MI190) **Année 2005–2006**

TD-TME 2

Machine de Krivine

TME Télécharger et exécuter le programme Caml suivant :
`http://www.pps.jussieu.fr/~emmanuel/Public/enseignement/2005-2006/ca/mk.ml`

TME Ajouter un mode debug pour la fonction interprète qui trace l'état de la pile, de l'environnement et du compteur ordinal. Tester la trace sur les exemples donnnés.

TME Ecrire un analyseur syntaxique pour le lambda-calcul acceptant les termes parenthésés et où

- l'application est associative à gauche
  ex : (A B C) correspond à ((A B) C)

et l'utiliser dans le programme.

TD Étendre le λ-calcul traité par le compilateur aux constantes entières et bolléennes, et aux déclarations locales `let in` et `let rec in`. Décrire l'évaluation de ces nouveaux termes, les instructions machines correspondantes, leur schéma de compilation et d'évaluation, et le nouveau compilateur et interprète associés.

TME Implanter ces modifications

Machine CAM

La mise en œuvre du langage Caml (l'ancêtre) repose sur la Machine Abstraite Catégorique (CAM en grand breton) . Cette machine peut être vue comme une synthèse des trois approches suivantes :

- le formalisme de De Bruijn qui élimine le nom des variables et résout les problèmes d'α-conversion.
- machine SK à réduction de Turner (les termes sont réécrits sous forme de combinateurs).
- machine SECD de Landin (machine à environnement et à pile).

Instructions de la machine

Chaque combinateur correspond à une instruction de la machine CAM, sauf la paire qui se décomposera en plusieurs instructions.
L'ensemble *Ins* des instructions de la machine est défini par :
Ins::=

Ins_F (instructions associées aux opérateurs de base),

Push : met l'accumulateur sur le sommet de pile,

Swap : échange le sommet de pile et l'accumulateur,

Cons : conse le sommet de pile à l'accumulateur,

Fst : prend le premier élément de l'accumulateur,

Snd : prend le deuxième élément de l'accumulateur,

Id : NOP

App : exécute le code de la fermeture du 1er élément de l'accumulateur

Cur : fabrique une fermeture (code + environnement)

Quote Val : n'évalue pas Val

Val peut prendre les valeurs :
Val::=

C : constante de base

Val : liste d'instructions (fermeture)

(Val,Val) : couple de valeurs

Schéma d'évaluation de la CAM

accumulateur	code	pile	accumulateur	code	pile
(s,t)	Fst;C	S	s	C	S
(s,t)	Snd;C	S	t	C	S
s	Id;C	S	s	C	S
s	Quote c;C	S	c	C	S
s	Cur (C);C1	S	$(s :: C)$	C1	S
s	Push;C	S	s	C	$s :: S$
t	Swap;C	$s :: S$	s	C	$t :: S$
t	Cons;C	$s :: S$	(s,t)	C	S
$(s :: C, t)$	App;C1	S	(s,t)	C;C1	S
s	Ins_f;C	S	$f(s)$	C	S

Traduction du λ-calcul en CAM

Le schéma de compilation utilise un motif p pour mémoriser la place des variables.

$$\begin{aligned}
[\![x]\!]_{(p,x)} &= Snd \\
[\![x]\!]_{(x,p)} &= Fst \\
[\![x]\!]_{(p_1,p_2)} &= (Snd; [\![x]\!]_{p_2})?(Fst; [\![x]\!]_{p_1}) \\
[\![MN]\!]_p &= < [\![M]\!]_p, [\![N]\!]_p >; App \\
[\![(M,N)]\!]_p &= < [\![M]\!]_p, [\![N]\!]_p > \\
[\![\lambda v.M]\!]_p &= Cur([\![M]\!]_{(p,v)}
\end{aligned}$$

La notation `e1 ? e2` signifie la valeur de `e1` si elle existe, sinon celle de `e2`.

Le combinateur paire donne la suite d'instructions suivantes :

$$< M, N > \rightarrow Push; M; Swap; N; Cons$$

L'exemple suivant montre la compilation du λ-terme $(\lambda x.xx)(\lambda x.x)$ et son évaluation avec les combinateurs catégoriques :

$$\begin{aligned}
[\![(\lambda x.xx)(\lambda x.x)]\!] &= < Cur(< Snd, Snd >; App), Cur(Snd) >; App \\
(Beta) &= < Id, Cur(Snd) >; (< Snd, Snd >; App) \\
(Ass)(Dpair)(Snd) &= < Cur(Snd), Cur(Snd) >; App \\
(Beta) &= < Id, Cur(Snd) >; Snd \\
(Snd) &= Cur(Snd) \\
[\![\lambda x.x]\!] &= Cur(Snd)
\end{aligned}$$

Le code CAM correspondant à $(\lambda x.xx)(\lambda x.x)$ est le suivant :

$$Push; Cur(M); Swap; Cur(N); App$$

où $M = xx\ (< Snd, Snd >; App)$ correspondant au code

$$Push; Snd; Swap; Snd; Cons; App$$

et $N = x\ (Snd)$ au code Snd.

L'exécution du code montre les différents états de l'environnement, du compteur ordinal et de la pile.

Accumulateur	*Code*	*Pile*
$()$	$Push; Cur(M); Swap; Cur(N); Cons; App$	$[]$
$()$	$Cur(M); Swap; Cur(N); Cons; App$	$[()]$
$(() : M)$	$Swap; Cur(N); Cons; App$	$[()]$
$()$	$Cur(N); Cons; App$	$[(() : M)]$
$(() : N)$	$Cons; App$	$[(() : M)]$
$((() : M), (() : N))$	App	$[]$
$((), (() : N))$	M	$[]$
$((), (() : N))$	$Push; Snd; Swap; Snd; Cons; App$	$[]$
$((), (() : N))$	$Snd; Swap; Snd; Cons; App$	$[((), (() : N))]$
$(() : N)$	$Swap; Snd; Cons; App$	$[((), (() : N))]$
$((), (() : N))$	$Snd; Cons; App$	$[(() : N)]$
$(() : N)$	$Cons; App$	$[(() : N)]$
$((() : N), (() : N))$	App	$[]$
$((), (() : N))$	N	$[]$
$((), (() : N))$	Snd	$[]$
$(() : N)$		$[]$

La machine s'arrête quand soit le compteur ordinal est vide, soit une instruction manipule le sommet de pile et celle-ci est vide. La valeur de l'évaluation est alors contenue dans le registre environnement. Celle-ci correspond bien au résultat escompté (N).

Représentation des environnements

Pour la CAM, les valeurs de l'environnement sont chaînées sous forme de liste. L'accès se fait par le numéro de la variable à rechercher. L'accès est légèrement plus long qu'un accès direct si l'environnement était sous forme de vecteur. Mais cela permet un plus grand partage des environnements en évitant des dupli cations inutiles.

A faire : implantation d'un interprète de la CAM

Le langage d'implantation est libre, mais essayer de le choisir dans un des langages suivants : O'CAML, C, JAVA.

TD Ecrire un compilateur de λ-termes vers la CAM.

TD Ecrire un interpréteur de la CAM.

TME Assembler ces deux parties avec un analyseur syntaxique de λ-calcul pour écrire un programme qui lit un λ-terme, le compile vers la CAM puis exécute le code produit en affichant la pile et l'environnement à chaque étape de calcul.