

The Categorical Abstract Machine

Hakjoo Oh `<pronto@ropas.snu.ac.kr>`

October 27, 2006

Weekly Show & Tell
Programming Research Laboratory,
Seoul National University
<http://ropas.snu.ac.kr/talk/2006/>



Contents

- Introduction
- The Abstract Machine
- Extension



- Introduction
- The Abstract Machine
- Extension



Goal

let $x = \text{plus}$ **in** $x(1,2)$

→ $(\lambda x.x(1,2)) \text{ plus}$

→ $(\lambda.0(1,2)) \text{ plus}$

→ $\text{App} \circ \langle \Lambda(\text{App} \circ \langle \text{Snd}, \langle '1', '2' \rangle \rangle), \Lambda(\text{plus} \circ \text{Snd}) \rangle$

→ $\text{push};(\text{cur push};\text{snd};\text{swap};\text{push};\text{quote } 1;\text{swap};\text{quote } 2;\text{cons};\text{cons};\text{app});\text{swap};(\text{cur snd};\text{plus});\text{cons};\text{app}$



S K I combinators

$$((\mathbf{K} x) y) = x$$

$$(\mathbf{S} x y z) = (xz (yz))$$

$$(\mathbf{I} x) = x (= (\mathbf{S} \mathbf{K} \mathbf{K}))$$



Abstraction Elimination

$$e ::= x \mid \lambda x.e \mid e_1 e_2$$

1. $T[x] \Rightarrow x$
2. $T[(e_1 e_2)] \Rightarrow (T[e_1] T[e_2])$
3. $T[\lambda x.e] \Rightarrow (\mathbf{KT}[e])$ if x is not free in e
4. $T[\lambda x.x] \Rightarrow \mathbf{I}$
5. $T[\lambda x.\lambda y.e] \Rightarrow T[\lambda x.T[\lambda y.e]]$ (if x is free in e)
6. $T[\lambda x.(e_1 e_2)] \Rightarrow (\mathbf{ST}[\lambda x.e_1] T[\lambda x.e_2])$



Example

let x = plus **in** x (4,(**let** x = 3 **in** 3));

M = $(\lambda x.x(4, (\lambda x.x) 3))_+$

N = $(\lambda x.x 4 ((\lambda x.x) 3))_+$

$T[\mathbf{N}] \Rightarrow \mathbf{S}(\mathbf{SI}(\mathbf{K4}))(\mathbf{S}(\mathbf{KI})(\mathbf{K3}))_+ = 7$



λ calculus in De Bruijn form

$$d ::= n \mid \lambda.d \mid d_1d_2$$

$$\|0\|(\rho, d) = d$$

$$\|n + 1\|(\rho, d) = \|n\|\rho$$

$$\|c\|\rho = c$$

$$\|MN\|\rho = \|M\|\rho(\|N\|\rho)$$

$$\|\lambda.M\|\rho d = \|M\|(\rho, d)$$

ρ has the shape of $(\dots ((\rho, v_n) \dots), v_0)$



Meanings with Combinators

$$\|n\| = n!, \|c\| = 'c, \|MN\| = S(\|M\|, \|N\|), \|\lambda.M\| = \Lambda(\|M\|)$$

$$0!(x, y) = y$$

$$(n + 1)!(x, y) = n!x$$

$$'x)y = x$$

$$S(x, y)z = xz(yz)$$

$$\Lambda(x)yz = x(y, z)$$



Pairing Combinator

$$\|(M, N)\| = \langle \|M\|, \|N\| \rangle$$

$$\|(M, N)\|\rho = (\|M\|\rho, \|N\|\rho)$$

$$Fst(x, y) = x$$

$$Snd(x, y) = y$$

$$\langle x, y \rangle z = (xz, yz)$$



Avoiding Redundancy

Let $S(,)$ and $n!$ as shorthands for $App \circ \langle, \rangle$ and $Snd \circ Fst^n$

$$(x \circ y)z = x(yz)$$

$$Fst(x, y) = x$$

$$Snd(x, y) = y$$

$$\langle x, y \rangle z = (xz, yz)$$

$$App(\Lambda(x)y, z) = x(y, z)$$

$$('x)y = x$$

for functional constants, like +

$$\Lambda(x \circ Snd)yz = (x \circ Snd)(y, z) = xz$$



Example (1/2)

let x = plus **in** x (4, (**let** x = 3 **in** 3));

$M = (\lambda x.x(4, (\lambda x.x) 3)) +$

$M' = S(\Lambda(S(0!, \langle '4, S(\Lambda(0!), '3 \rangle)), \Lambda(+ \circ Snd))$

$A = S(0!, \langle '4, B \rangle)$

$B = S(\Lambda(0!), '3)$



Example (2/2)

$$\begin{aligned}
 & S(\Lambda(A), \Lambda(+ \circ Snd))() \rightarrow App(\Lambda(A)(), \Lambda(+ \circ Snd)()) \\
 & \rightarrow A\rho(\rho = ((), \Lambda(+ \circ Snd)())) \rightarrow App(0!\rho, \langle '4, B \rangle \rho) \\
 & \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4, \mathbf{B} \rangle \rho) \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4\rho, B\rho \rangle) \\
 & \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4, \mathbf{B}\rho \rangle) \\
 & \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4, App(\Lambda(0)\rho, '3\rho) \rangle) \\
 & \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4, \mathbf{App}(\Lambda(0!\rho), 3) \rangle) \\
 & \rightarrow App(\Lambda(+ \circ Snd)(), \langle '4, 0!(\rho, 3) \rangle) \rightarrow App(\Lambda(+ \circ \mathbf{Snd})(), \langle '4, 3 \rangle) \\
 & \rightarrow (+ \circ Snd)(), (4, 3) \rightarrow +(Snd(), (4, 3)) \rightarrow +(4, 3) \rightarrow 7
 \end{aligned}$$



- Introduction
- The Abstract Machine
- Extension



Translation to Machine Code (1/3)

Machine Configuration = $\langle \text{term, code, stack} \rangle$

- ▶ $\underline{x \circ y} = \underline{yx}$
- ▶ $\underline{fst} = fst$: expects a term (s, t) and replaces it by s
- ▶ $\underline{snd} = snd$: expects a term (s, t) and replaces it by t
- ▶ $\underline{c} = (\text{quote } c)$

Term	Code	Stack	Term	code	Stack
(s, t)	$fst; C$	S	s	C	S
(s, t)	$snd; C$	S	t	C	S
s	$(\text{quote } c); C$	S	c	C	S



Translation to Machine Code (2/3)

Machine Configuration = $\langle \text{term}, \text{code}, \text{stack} \rangle$

- ▶ $\langle \underline{M}, \underline{N} \rangle = \text{push}; \underline{M}; \text{swap}; \underline{N}; \text{cons}$
 - *push*: push the term onto the top of the stack
 - *swap*: swap the term and the top of the stack
 - *cons*: make a couple out of the top of the stack and the term, replace the term by the couple just built, and pop the stack

Term	Code	Stack	Term	code	Stack
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



Translation to Machine Code (3/3)

Machine Configuration = $\langle \text{term}, \text{code}, \text{stack} \rangle$

- ▶ $\underline{\Lambda}(M) = (\text{cur } \underline{M})$
 - *cur*: replace the term s by $C : s$ where C is in the code encapsulated in Λ
- ▶ $\underline{App} = \text{app}$
 - *app*: expects a term $(C : s, t)$, replaces it by (s, t) and prefixes the rest of the code by C .
- ▶ $\underline{+} = \Lambda(\text{snd}; \text{plus})$ for functional constants

Term	Code	Stack	Term	code	Stack
s	$(\text{cur } C); C1$	S	$(C:s)$	$C1$	S
$(C:s,t)$	$\text{app}; C1$	S	(s,t)	$C; C1$	S
(m,n)	$\text{plus}; C$	S	$m+n$	C	S



Running Example

let $x = \text{plus}$ **in** $x(1,2)$
 $\rightarrow (\lambda x.x(1,2)) \text{ plus}$
 $\rightarrow (\lambda.0(1,2)) \text{ plus}$
 $\rightarrow \text{App} \circ \langle \Lambda(\text{App} \circ \langle \text{Snd}, \langle '1, '2 \rangle \rangle), \Lambda(\text{plus} \circ \text{Snd}) \rangle$
 $\rightarrow \text{push};(\text{cur push};\text{snd};\text{swap};\text{push};\text{quote } 1;\text{swap};\text{quote } 2;\text{cons};\text{cons};\text{app});\text{swap};(\text{cur snd};\text{plus});\text{cons};\text{app}$

Term	Code	Stack
$()$	$\text{push};\text{cur } C1;\text{swap};\text{cur } C2;\text{cons};\text{app}$	$[]$
$C1:()$	$\text{swap};\text{cur } C2;\text{cons};\text{app}$	$[()]$
$C2:()$	$\text{cons};\text{app}$	$[C1:()]$
$(C1:(),C2:())$	app	$[]$
$((),C2:())$	$C1$	$[]$
$(\text{snd};\text{plus}:(), (1,2))$	app	$[]$
$((), (1,2))$	$\text{snd};\text{plus}$	$[]$
$(1,2)$	plus	$[]$



Optimization - excluding couple, app (1/3)

$$(\lambda.M) N$$

Term	Code	Stack
s	push;cur C;swap;C1;cons;app	S
s	cur C;swap;C1;cons;app	s.S
C:s	swap;cur C1;cons;app	s.S
s	C1;cons;app	(C:s).S
v	cons;app	(C:s).S
(C:s,v)	app	S
(s,v)	C	S



Optimization - excluding couple, app (2/3)

$$(\lambda.M) N$$

Term	Code	Stack
s	push;C1;cons;C	S
s	C1;cons;C	s.S
v	cons;C	s.S
(s,v)	C	S



Optimization - excluding couple, app (3/3)

Natural β -reduction using identity combinator

$$App \circ \langle \Lambda(x), y \rangle = x \circ \langle Id, y \rangle$$

$$Fst \circ \langle x, y \rangle = x$$

$$Snd \circ \langle x, y \rangle = y$$

let x = N in M

Term	Code	Stack
s	push;skip;swap;C1;cons;C	S
s	skip;swap;C1;cons;C	s.S
s	swap;C1;cons;C	s.S
s	C1;cons;C	s.S
v	cons;C	s.S
(s,v)	C	S



- Introduction
- The Abstract Machine
- Extension

Conditionals

if M then N else P = push;M;branch(N,P)

branch : replace the term by the top of the stack, and, according to whether the term is true or false, execute N or P

Term	Code	Stack	Term	code	Stack
true	branch(C1,C2);C	s.S	s	C1;C	S
false	branch(C1,C2);C	s.S	s	C2;C	S



Recursion (1/3)

using Y-combinator,

$$YM = M(YM) \Leftrightarrow \llbracket YM \rrbracket = \text{App} \circ \langle \llbracket M \rrbracket, \llbracket YM \rrbracket \rangle$$

introduce unary combinator *Fix*, where *Fix*(*C*) is the abbreviation of $\text{App} \circ \langle \llbracket Y \rrbracket, C \rangle$

$$\text{Fix}(C) = \text{App} \circ \langle C, \text{Fix}(C) \rangle$$

assume that the argument of *Fix* has a form $\Lambda(\Lambda(M))$

$$\text{Fix}(\Lambda(\Lambda(M))) = \text{App} \circ \langle \Lambda(\Lambda(M)), \text{Fix}(\Lambda(\Lambda(M))) \rangle = \Lambda(M) \circ \langle \text{Id}, \text{Fix}(\Lambda(\Lambda(M))) \rangle$$

abbreviating $\text{Fix}(\Lambda(\Lambda(M)))$ to *F*(*M*)

$$F(M) = \Lambda(M) \circ \langle \text{Id}, F(M) \rangle$$



Recursion (2/3)

$$\begin{aligned} F(M) &= \Lambda(M) \circ \langle Id, F(M) \rangle \\ &= \underline{push; F(\underline{M}); cons; (cur \underline{M})} \end{aligned}$$

Let t be the result of the action of $F(C)$ on the term s .

$$t = C : (s, t)$$

decompose the effect of $F(C)$ into

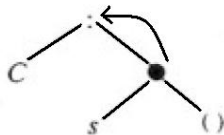
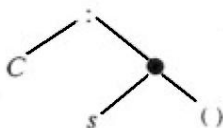
1. construct an object $C : (s, ())$, where $()$ is dummy value
2. construct the looping structure through a new instruction *wind*

Recursion (3/3)

Term	Code	Stack	Term	code	Stack
s	wind;C	(t,()).S	s[(t,()) ← (t,s)]	C	S

for example,

Term	Code	Stack
B:((),())	w	[(() , ()) ; ()]
B:(s=((),B:s))		[()]



Lazy Evaluation

- ▶ freeze: replace the term s by the structure $(C.s)$, where C is the code encapsulated in the freeze instruction
- ▶ unfreeze: performs no action unless the term is a lazy $C.s$, in which case C is prefixed to the code (including unfreeze) and the term becomes s .
 - the compiler has to insert 'unfreeze' right before 'strict' instructions, i.e. those which cannot be executed on a lazy: $\text{fst}, \text{snd}, \text{app}, \text{plus}$.

Term	Code	Stack	Term	code	Stack
s	$(\text{freeze } C); C1$	S	$C.s$	$C1$	S
$C.s$	$\text{unfreeze}; C1$	S	s	$C; \text{unfreeze}; C1$	S
s	$\text{unfreeze}; C$	S	s	C	S



Example of Lazy Evaluation

letrec $x = (1, \text{freeze } x)$ **in** $\text{fst}(\text{snd}(x))$

The compiler translates $\text{fst}(M)$ by $\text{fst}(\text{unfreeze}(M))$



References



G.Cousineau, P.L.Curien and M.Mauny
The Categorical Abstract Machine
Science of Computer Programming 8, 1987