The Categorical Abstract Machine

Kesha Hietala

Source Language

- t ::= *x* ; variable
  | *c* ; constant
  | λ*x*.t ; abstraction
  | (t t) ; application
  | (t, t) ; pair

- Can add features like conditionals and data constructors later

Complexity of Computation

- functions can result from computation

- constructed functions may require environments

```
fun f x =                    val x = (f 2)
    let g y = x + y          val y = (f 3)
    in g
```

- Solution is to use closures

Evaluation Model

- eval: expr * env -> expr

- Meanings of expressions:

  $[\!| x |\!]\ \rho = \rho(x)$

  $[\!| c |\!]\ \rho = c$

  $[\!| (M\ N) |\!]\ \rho = [\!| M |\!]\ \rho\ ([\!| N |\!]\ \rho)$

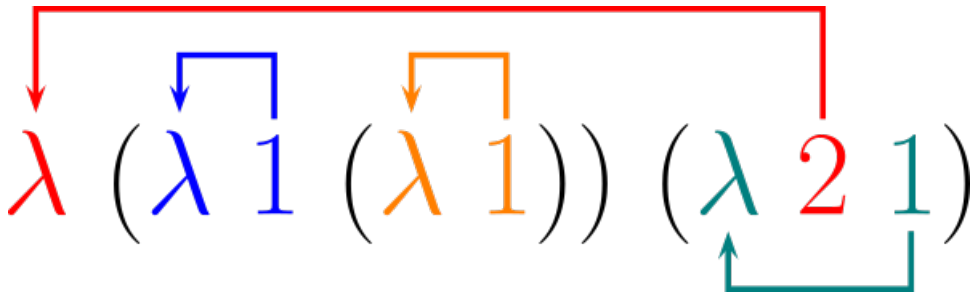
  $[\!| \lambda x.M |\!]\ \rho d = [\!| M |\!]\ \rho[x \leftarrow d]$

  $[\!| (M,N) |\!]\ \rho = ([\!| M |\!]\ \rho, [\!| N |\!]\ \rho)$

De Bruijn Form

- Idea is to replace each name with a number recording the variable's binding height

- e.g. λ*z*. (λ*y*. *y* (λ*x*. *x*)) (λ*x*. *z x*) becomes:



*source: http://en.wikipedia.org/wiki/De_Bruijn_index

Modified Evaluation Model

- Using De Bruijn indices the environment becomes a simple list of values

$[\!|\ 1\ |\!]\ (\rho,d) = d \qquad [\!|\ n{+}1\ |\!]\ (\rho,d) = [\!|\ n\ |\!]\ \rho$

$[\!|\ c\ |\!]\ \rho = c$

$[\!|\ (M\ N)\ |\!]\ \rho = [\!|\ M\ |\!]\ \rho\ ([\!|\ N\ |\!]\ \rho)$

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

$[\!|\ (M,N)\ |\!]\ \rho = ([\!|\ M\ |\!]\ \rho, [\!|\ N\ |\!]\ \rho)$

CAM Combinators

- Introduce a set of constants to encode meaning rules: Λ, Fst, Snd, $\circ$, <, >, ‘, App

- Rules for evaluation:

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

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

$\text{Fst}\ (x, y) = x$

$\text{App}(\Lambda(x)\ y, z) = x\ (y, z)$

$\text{Snd}\ (x, y) = y$

$(‘x)\ y = x$

Translation into Combinatory Form

- Translation rules:

  $\mathbf{T}(1) = \mathrm{Snd}$

  $\mathbf{T}(n+1) = \mathbf{T}(n) \circ \mathrm{Fst}$

  $\mathbf{T}(c) = `c$

  $\mathbf{T}(\ (M\ N)\ ) = \mathrm{App} \circ \langle \mathbf{T}(M), \mathbf{T}(N) \rangle$

  $\mathbf{T}(\lambda.M) = \Lambda(\mathbf{T}(M))$

- example:

$(\lambda x.+(1,x))\ 2 \rightarrow \mathrm{App} \circ \langle \Lambda(\mathrm{App} \circ \langle +, \langle `1, \mathrm{Snd} \rangle \rangle), `2 \rangle$

CAM Model

- Consider evaluation of an application (t_1 t_2)
  1. save environment *e*
  2. evaluate t_1 to t_1'
  3. save t_1' and restore *e*
  4. evaluate t_2 to t_2'
  5. apply t_1' to t_2'

- This suggests a model using a term, code, and stack

CAM Instructions

- Goal: transform combinatory expressions into code for the CAM model

- A few examples:
  - App $\circ$ $\langle t_1, t_2 \rangle$ -> [push, t_1^C, swap, t_2^C, cons, app]
  - $t_1 \circ t_2$ -> [t_2^C, t_1^C]
  - $\Lambda(t)$ -> [cur [t^C]]

Instruction Operational Semantics

Term	Code	Stack
(s, t)	fst;C	S
(s, t)	snd;C	S
s	(quote c);C	S
s	(cur C);C1	S
s	push;C	S
t	swap;C	s.S
t	cons;C	s.S
(C:s, t)	app;C1	S
(m, n)	plus;C	S

Term	Code	Stack
s	C	S
t	C	S
c	C	S
(C:s)	C1	S
s	C	s.S
s	C	t.S
(s, t)	C	S
(s, t)	C;C1	S
m + n	C	S

Translation to Code

T(App) = [app]

T(*M* ∘ *N*) = **T**(*N*) + **T**(*M*)

T(Snd) = [snd]

T(Fst) = [fst]

T(‘c) = [quote c]

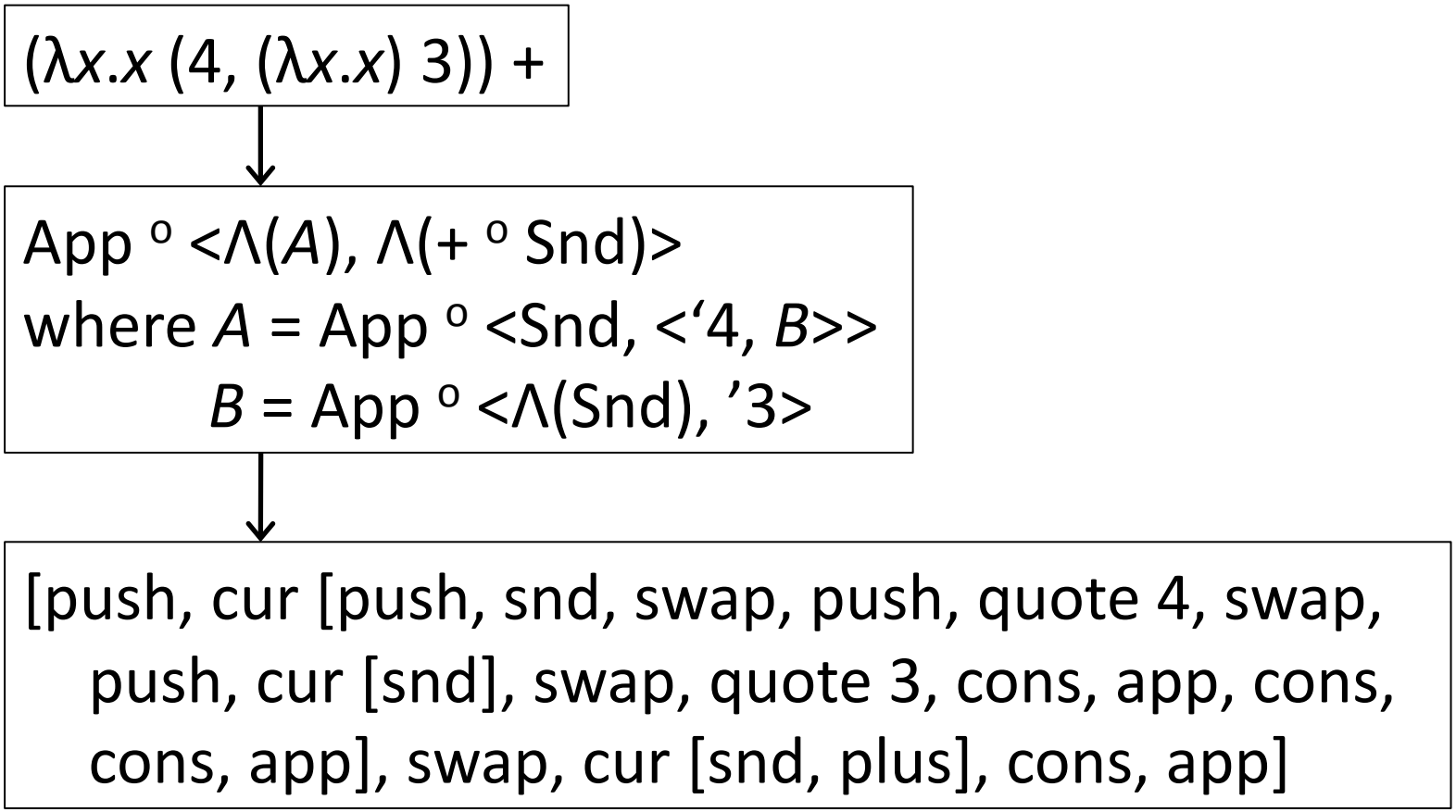
T(Λ(*M*)) = [cur(**T**(*M*))]

T(<*M*, *N*>) = [push] + **T**(*M*) + [swap] + **T**(*N*) + [cons]

T(+) = [plus]

Example

- **let** *x* = + **in** *x* (4, (*x* **where** *x* = 3))



Adding Conditionals

- branch(C_1, C_2):
  - Depending on whether the term is true or false, replace it with the environment at the top of the stack and execute C_1 or C_2

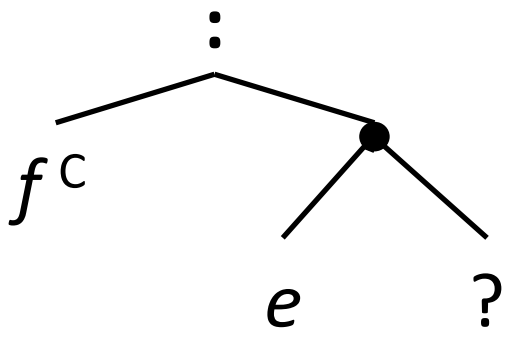
- **if** t_1 **then** t_2 **else** t_3 ->

  [push, t_1^C, branch(t_2^C, t_3^C)]

Recursion

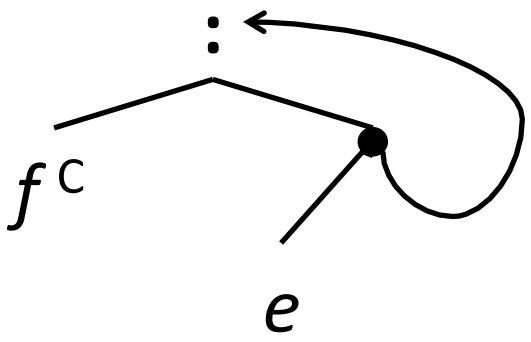
- e.g. **letrec** *f x* = ... (*f* 1)... **in** ...
- Need to add some definition for *f* to the environment before evaluating the function body

[push, ?, cons, cur (f^{C})]



Recursion (cont.)

[push, ?, cons, push, cur (f^{C}), wind]



Term	Code	Stack
s	wind;C	(t,?).S

Term	Code	Stack
s[(t,?) <- (t,s)]	C	S

Factorial Example

- **letrec** *fact n* =
  if *n* = 0 **then** 1 **else** *n* * *fact* (*n* – 1)
  in (*fact* 1)

[push, push, **unit**, cons, push, cur *A*, **wind**, cons, push, snd, swap,
quote 1, cons, app]

A = [push, push, cur [snd, equals], swap, push, snd, swap,
quote 0, cons, cons, app, branch ([quote 1], *B*)

B = [push, cur [snd, times], swap, push, snd, swap, push, fst, snd,
swap, push, cur [snd, minus], swap, push, snd, swap, quote 1,
cons, cons, app, cons, app, cons, cons, app]

Questions?