The Categorical Abstract Machine

Kesha Hietala
Source Language

- $t ::= x$ ; variable
- $c$ ; constant
- $\lambda x.t$ ; 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:
  
  \[ \lfloor \lfloor x \rfloor \rfloor \rho = \rho(x) \]
  
  \[ \lfloor \lfloor c \rfloor \rfloor \rho = c \]
  
  \[ \lfloor \lfloor (M N) \rfloor \rfloor \rho = \lfloor \lfloor M \rfloor \rfloor \rho (\lfloor \lfloor N \rfloor \rfloor \rho) \]
  
  \[ \lfloor \lfloor \lambda x.M \rfloor \rfloor \rho d = \lfloor \lfloor M \rfloor \rfloor \rho [x <- d] \]
  
  \[ \lfloor \lfloor (M, N) \rfloor \rfloor \rho = (\lfloor \lfloor M \rfloor \rfloor \rho, \lfloor \lfloor N \rfloor \rfloor \rho) \]
De Bruijn Form

• Idea is to replace each name with a number recording the variable’s binding height

• e.g. $\lambda z. (\lambda y. y (\lambda x. x)) (\lambda x. z x)$ 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

\[
\begin{align*}
\langle 1, d \rangle &= d & \langle n+1, d \rangle &= \langle n, \rho \rangle \\
\langle c, \rho \rangle &= c \\
\langle (M, N), \rho \rangle &= \langle M, \rho \rangle \langle N, \rho \rangle \\
\langle \lambda M, d \rangle &= \langle M, \rho(d) \rangle \\
\langle (M, N), \rho \rangle &= \langle M, \rho \rangle, \langle N, \rho \rangle 
\end{align*}
\]
CAM Combinators

• Introduce a set of constants to encode meaning rules: $\wedge$, Fst, Snd, $\circ$, $<$, $>$, $'$, App

• Rules for evaluation:

  $(x \circ y) z = x (yz)$  \hspace{1cm} <x, y> z = <xz, yz>

  Fst (x, y) = x  \hspace{1cm} App (\wedge (x) y, z) = x (y, z)

  Snd (x, y) = y  \hspace{1cm} ('x) y = x$
Translation into Combinatory Form

• Translation rules:
  \[ T(1) = \text{Snd} \]
  \[ T(n+1) = T(n) \circ \text{Fst} \]
  \[ T(c) = 'c \]
  \[ T( (M \ N) ) = \text{App} \circ \langle T(M), T(N)\rangle \]
  \[ T(\lambda.M) = \lambda(T(M)) \]

• example:
  \[ (\lambda x.+(1,x)) \ 2 \rightarrow \ \text{App} \circ \langle \lambda(\text{App} \circ <+,'1,\text{Snd}\rangle), '2\rangle \]
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:
  – $\text{App} \circ <t_1,t_2> \rightarrow [\text{push}, t_1^C, \text{swap}, t_2^C, \text{cons}, \text{app}]
  – t_1 \circ t_2 \rightarrow [t_2^C, t_1^C]
  – \Lambda(t) \rightarrow [\text{cur } [t^C]]$
### Instruction Operational Semantics

<table>
<thead>
<tr>
<th>Term</th>
<th>Code</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s, t)</td>
<td>fst;C</td>
<td>S</td>
</tr>
<tr>
<td>(s, t)</td>
<td>snd;C</td>
<td>S</td>
</tr>
<tr>
<td>s</td>
<td>(quote c);C</td>
<td>S</td>
</tr>
<tr>
<td>s</td>
<td>(cur C);C1</td>
<td>S</td>
</tr>
<tr>
<td>s</td>
<td>push;C</td>
<td>S</td>
</tr>
<tr>
<td>t</td>
<td>swap;C</td>
<td>s.S</td>
</tr>
<tr>
<td>t</td>
<td>cons;C</td>
<td>s.S</td>
</tr>
<tr>
<td>(C:s, t)</td>
<td>app;C1</td>
<td>S</td>
</tr>
<tr>
<td>(m, n)</td>
<td>plus;C</td>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Code</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>C</td>
<td>S</td>
</tr>
<tr>
<td>t</td>
<td>C</td>
<td>S</td>
</tr>
<tr>
<td>c</td>
<td>C</td>
<td>S</td>
</tr>
<tr>
<td>(C:s)</td>
<td>C1</td>
<td>S</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>s.S</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>t.S</td>
</tr>
<tr>
<td>(s, t)</td>
<td>C</td>
<td>S</td>
</tr>
<tr>
<td>(s, t)</td>
<td>C;C1</td>
<td>S</td>
</tr>
<tr>
<td>m + n</td>
<td>C</td>
<td>S</td>
</tr>
</tbody>
</table>
Translation to Code

\[ T(\text{App}) = [\text{app}] \]
\[ T(M \circ N) = T(N) + T(M) \]
\[ T(\text{Snd}) = [\text{snd}] \]
\[ T(\text{Fst}) = [\text{fst}] \]
\[ T(\text{‘c}) = [\text{quote c}] \]
\[ T(\Lambda(M)) = [\text{cur}(T(M))] \]
\[ T(<M, N>) = [\text{push}] + T(M) + [\text{swap}] + T(N) + [\text{cons}] \]
\[ T(+) = [\text{plus}] \]
Example

• let $x = +$ in $x (4, (x \text{ where } x = 3))$

\[
(\lambda x. x (4, (\lambda x. x) 3)) +
\]

App $^\circ <\Lambda(A), \Lambda(+ ^\circ \text{Snd})>$
where $A = \text{App}^\circ <\text{Snd}, <'4, B>>$
\[
B = \text{App}^\circ <\Lambda(\text{Snd}), '3>
\]

[push, cur [push, snd, swap, push, push, quote 4, swap, push, cur [snd], swap, quote 3, cons, app, cons, cons, app], swap, cur [snd, plus], cons, app]
Adding Conditionals

• branch(C₁, C₂):
  – Depending on whether the term is true or false, replace it with the environment at the top of the stack and execute C₁ or C₂

• if t₁ then t₂ else t₃ →
  [push, t₁^C, branch(t₂^C, t₃^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

$[\text{push, }?, \text{cons, cur } (f^C)]$
Recursion (cont.)

[push, ?, cons, push, cur \((f^C)\), wind]
Factorial Example

- **letrec** `{fact n =}
  
  if \( n = 0 \) then 1 else \( n \times fact \ (n - 1) \)
  
  in \( fact \ (n + 1 \)  

\[
\begin{align*}
A &= \text{push, push, } \text{unit, cons, push, cur } A, \ \text{wind, cons, push, snd, swap,}
\quad \text{quote } 1, \ \text{cons, app, branch ( } \text{quote } 1, \ B) \\
B &= \text{push, cur [snd, times], swap, push, snd, swap, push, fst, snd,}
\quad \text{swap, push, cur [snd, minus], swap, push, snd, swap, quote } 1,
\quad \text{cons, cons, app, app, cons, app, cons, cons, app}
\end{align*}
\]
Questions?