Converting Combinatory Logic to and from Concatenative Calculus

Daniel Kiyoshi Hashimoto
Hugo Musso Gualandi

Instituto de Computação – Universidade Federal do Rio de Janeiro

September 2024

INSTITUTO DE COMPUTAÇÃO UFRJ

Tacit Programming

Programming without named variables.

Normal	Tacit	
$\lambda x \,.\, f\ (g\ x)$	$f \circ g$	Functional
$\lambda x\, y \,.\, (x {*} y) + 1$	$* \; 1 \; +$	Stack
	No x, y here!	

What is that doing?

Functional style (`Haskell`):

$$(\text{flip} \circ \text{const})\ \text{id}$$

Stack style (`Joy`):

`swap zap`

`Haskell`

$$\begin{aligned} (f \circ g)\ x &\to f\ (g\ x) \\ \text{flip}\ f\ x\ y &\to f\ y\ x \\ \text{const}\ a\ b &\to a \\ \text{id}\ x &\to x \end{aligned}$$

`Joy`

$$\begin{aligned} y\ x\ \texttt{swap} &\to x\ y \\ x\ \texttt{zap} &\to \varepsilon \end{aligned}$$

INSTITUTO DE COMPUTAÇÃO UFRJ

What is that doing?

Functional style (`Haskell`):

$$(\text{flip} \circ \text{const})\ \text{id}\ x\ y$$

Stack style (`Joy`):

$$y\ x\ \texttt{swap}\ \texttt{zap}$$

`Haskell`

$$\begin{aligned} (f \circ g)\ x &\rightarrow f\ (g\ x) \\ \text{flip}\ f\ x\ y &\rightarrow f\ y\ x \\ \text{const}\ a\ b &\rightarrow a \\ \text{id}\ x &\rightarrow x \end{aligned}$$

`Joy`

$$\begin{aligned} y\ x\ \texttt{swap} &\rightarrow x\ y \\ x\ \texttt{zap} &\rightarrow \varepsilon \end{aligned}$$

What is that doing?

Functional style (`Haskell`):

$$(\text{flip} \circ \text{const})\ \text{id}\ x\ y$$
$$\text{flip}\ (\text{const id})\ x\ y$$
$$\text{const id}\ y\ x$$
$$\text{id}\ x$$
$$x$$

Stack style (`Joy`):

$$y\ x\ \texttt{swap}\ \texttt{zap}$$
$$x\ y\ \texttt{zap}$$
$$x$$

`Haskell`

$$\begin{aligned}(f \circ g)\ x &\rightarrow f\ (g\ x)\\ \text{flip}\ f\ x\ y &\rightarrow f\ y\ x\\ \text{const}\ a\ b &\rightarrow a\\ \text{id}\ x &\rightarrow x\end{aligned}$$

`Joy`

$$\begin{aligned}y\ x\ \texttt{swap} &\rightarrow x\ y\\ x\ \texttt{zap} &\rightarrow \varepsilon\end{aligned}$$

INSTITUTO DE COMPUTAÇÃO UFRJ

What is that doing?

Functional style (`Haskell`):

$$(\text{flip} \circ \text{const})\ \text{id}\ x\ y$$
$$\text{flip}\ (\text{const id})\ x\ y$$
$$\text{const id}\ y\ x$$
$$\text{id}\ x$$
$$x$$

$\Longleftrightarrow$

Stack style (`Joy`):

$$y\ x\ \texttt{swap}\ \texttt{zap}$$
$$x\ y\ \texttt{zap}$$
$$x$$

Haskell

$$\begin{aligned} (f \circ g)\ x &\to f\ (g\ x) \\ \text{flip}\ f\ x\ y &\to f\ y\ x \\ \text{const}\ a\ b &\to a \\ \text{id}\ x &\to x \end{aligned}$$

Joy

$$\begin{aligned} y\ x\ \texttt{swap} &\to x\ y \\ x\ \texttt{zap} &\to \varepsilon \end{aligned}$$

INSTITUTO DE COMPUTAÇÃO UFRJ

What is that doing?

Functional style (`Haskell`): Stack style (`Joy`):

$$(\text{flip} \circ \text{const})\ \text{id}\ x\ y \iff y\ x\ \texttt{swap}\ \texttt{zap}$$

$$\text{flip}\ (\text{const id})\ x\ y$$

$$\text{const id}\ y\ x \iff x\ y\ \texttt{zap}$$

$$\text{id}\ x$$

$$x \qquad x$$

`Haskell`

$$\begin{aligned} (f \circ g)\ x &\rightarrow f\ (g\ x) \\ \text{flip}\ f\ x\ y &\rightarrow f\ y\ x \\ \text{const}\ a\ b &\rightarrow a \\ \text{id}\ x &\rightarrow x \end{aligned}$$

`Joy`

$$\begin{aligned} y\ x\ \texttt{swap} &\rightarrow x\ y \\ x\ \texttt{zap} &\rightarrow \varepsilon \end{aligned}$$

What is that doing?

Functional style (`Haskell`): | Stack style (`Joy`):

$$\begin{array}{lcl}
(\text{flip} \circ \text{const})\ \text{id}\ x\ y & \Longleftrightarrow & y\ x\ \texttt{swap}\ \texttt{zap} \\
\text{flip}\ (\text{const id})\ x\ y & & \\
\text{const id}\ y\ x & \Longleftrightarrow & x\ y\ \texttt{zap} \\
\text{id}\ x & & \\
x & \Longleftrightarrow & x
\end{array}$$

`Haskell`

$$\begin{array}{lcl}
(f \circ g)\ x & \rightarrow & f\ (g\ x) \\
\text{flip}\ f\ x\ y & \rightarrow & f\ y\ x \\
\text{const}\ a\ b & \rightarrow & a \\
\text{id}\ x & \rightarrow & x
\end{array}$$

`Joy`

$$\begin{array}{rcl}
y\ x\ \texttt{swap} & \rightarrow & x\ y \\
x\ \texttt{zap} & \rightarrow & \varepsilon
\end{array}$$

What is that doing?

Functional style (`Haskell`): Stack style (`Joy`):

Functional style		Stack style
$(\mathrm{flip} \circ \mathrm{const})\ \mathrm{id}\ x\ y$	$\Longleftrightarrow$	$y\ x$ `swap zap`
$\mathrm{flip}\ (\mathrm{const}\ \mathrm{id})\ x\ y$		
$\mathrm{const}\ \mathrm{id}\ y\ x$	$\Longleftrightarrow$	$x\ y$ `zap`
$\mathrm{id}\ x$		
x	$\Longleftrightarrow$	x

- Stack simulates functional intuitively.
- The steps matter: `swap swap` $\neq$ `noop`.

Goal

The problem:

- Functional: Well understood but harder to read
- Higher-order stack: Intuitive but less studied

The solution is to describe simulations:

- Functional $\rightarrow$ Higher-order stack: gain intuition
- Higher-order stack $\rightarrow$ Functional: inherit formalization

The Concatenative Calculus

A model for higher-order stack languages.

Concatenation is composition!

Instructions work on values to its left:

$$y\ x\ \texttt{swap}\ \texttt{zap}\ \mapsto\ x\ y\ \texttt{zap}\ \mapsto\ x$$

A quotation is an anonymous block of code:

$$y\ x\ [\,\texttt{swap}\,]\ \texttt{call}\ \mapsto\ y\ x\ \texttt{swap}\ \mapsto\ x\ y$$

`dip` extends the reach of other programs:

$$y\ x\ [\,\texttt{zap}\,]\ \texttt{dip}\ \mapsto\ y\ \texttt{zap}\ x\ \mapsto\ x$$

Thun (1994), Kerby (2002), Kleffner (2017).

INSTITUTO DE COMPUTAÇÃO UFRJ

Combinatory Logic

A tacit model for functional programming.

Single letter combinators:

$$
\begin{array}{ccc}
\textit{permute} & C\ f\ x\ y & \longrightarrow & f\ y\ x \\
\textit{duplicate} & W\,f\ x & \longrightarrow & f\ x\ x \\
\textit{discard} & K\ x\ y & \longrightarrow & x \\
\textit{identity} & I\ \ x & \longrightarrow & x \\
\textit{compose} & B\ f\ g\ x & \longrightarrow & f\ (g\ x) \\
\textit{split} & S\ f\ g\ x & \longrightarrow & f\ x\ (g\ x)
\end{array}
$$

Created by Schönfinkel (1924); expanded and popularized by Curry (1930).

INSTITUTO DE COMPUTAÇÃO UFRJ

Combinatory Logic

A tacit model for functional programming.

Single letter combinators:

$$
\begin{array}{lllll}
\textit{permute} & C\ f\ x\ y & \longrightarrow & f\ y\ x & \textit{flip} \\
\textit{duplicate} & W\,f\ x & \longrightarrow & f\ x\ x & \\
\textit{discard} & K\ x\ y & \longrightarrow & x & \textit{const} \\
\textit{identity} & I\ x & \longrightarrow & x & \textit{id} \\
\textit{compose} & B\ f\ g\ x & \longrightarrow & f\ (g\ x) & \circ \\
\textit{split} & S\ f\ g\ x & \longrightarrow & f\ x\ (g\ x) &
\end{array}
$$

Created by Schönfinkel (1924); expanded and popularized by Curry (1930).

Regular combinators

Continuation-Passing Style!

- calling the continuation is returning

$$\begin{array}{ccl} \textit{permute} & C\ q\ x\ y & \longrightarrow\ q\ y\ x \\ \textit{duplicate} & W\, q\ x & \longrightarrow\ q\ x\ x \\ \textit{discard} & K\ q\ x & \longrightarrow\ q \\ \textit{identity} & I\ \ q & \longrightarrow\ q \\ \textit{compose} & B\ q\ f\ x & \longrightarrow\ q\ (f\ x) \\ \textit{split} & S\ q\ f\ x & \longrightarrow\ q\ x\ (f\ x) \end{array}$$

$B\,C\,K$ is regular:

$$B\,C\,K\,q\,x\,y \longrightarrow C\,(K\,q)\,x\,y \longrightarrow K\,q\,y\,x \longrightarrow q\,x$$

$C\,I$ is not regular:

$$C\,I\,q\,x \longrightarrow I\,x\,q \longrightarrow x\,q$$

B is not only composition!

B is interpreted in different ways, depending on where q is.

B with 0 arguments is application:

$\underline{B}\ q\ f\ x \longrightarrow q\ (f\ x)$ `x [P] call`

B with 1 argument does stashing:

$\underline{B\ f}\ q\ x \longrightarrow f\ (q\ x)$ `x [P] dip`

B with 2 arguments composes:

$\underline{B\ f\ g}\ q \longrightarrow f\ (g\ q)$ `P Q`

INSTITUTO DE COMPUTAÇÃO UFRJ

C is `swap`!

Each combinator usage matches one instruction:

$$
\begin{array}{rcl c rcl}
C\ q\ x\ y & \longrightarrow & q\ y\ x & \textit{permute} & y\ x\ \texttt{swap} & \mapsto & x\ y \\
W\ q\ x & \longrightarrow & q\ x\ x & \textit{duplicate} & x\ \texttt{dup} & \mapsto & x\ x \\
K\ q\ x & \longrightarrow & q & \textit{discard} & x\ \texttt{zap} & \mapsto & \varepsilon \\
I\ q & \longrightarrow & q & \textit{identity} & \varepsilon & & \\
B\ q\ f\ x & \longrightarrow & q\ (f\ x) & \textit{apply} & x\ [\ \texttt{P}\ ]\ \texttt{call} & \mapsto & x\ \texttt{P} \\
B\ f\ q\ x & \longrightarrow & f\ (q\ x) & \textit{stash} & x\ [\ \texttt{P}\ ]\ \texttt{dip} & \mapsto & \texttt{P}\ x \\
B\ f\ g\ q & \longrightarrow & f\ (g\ q) & \textit{composition} & \texttt{P}\ \texttt{Q} & &
\end{array}
$$

It is a simulation!

Once q is nested inside, the evaluation happens in lockstep:

$$
\begin{array}{rcl}
B\,(B\,K\,C)\,W\,q\,x\,y\,z & \Leftrightarrow & z\;y\;x\;\texttt{zap swap dup} \\
B\,K\,C\,(W\,q)\,x\,y\,z & \Leftrightarrow & z\;y\;x\;\texttt{zap swap dup} \\
K\,(C\,(W\,q))\,x\,y\,z & \Leftrightarrow & z\;y\;x\;\texttt{zap swap dup} \\
C\,(W\,q)\,y\,z & \Leftrightarrow & z\;y\;\texttt{swap dup} \\
W\,q\,z\,y & \Leftrightarrow & y\;z\;\texttt{dup} \\
q\,z\,z\,y & \Leftrightarrow & y\;z\;z
\end{array}
$$

In concatenative programs, the continuation is implicit.

INSTITUTO DE COMPUTAÇÃO UFRJ

Regular-ish and Higher-Order

Until now, `dip` had to be with a quotation (like for-loops).
We want to decouple `dip` from the quotation (like map).

pushing α:

$$\begin{array}{rcl} \underline{C\,I\,\alpha}\;q & \Leftrightarrow & \texttt{[P]} \\ I\;q\;\alpha & & \\ q\;\alpha & \Leftrightarrow & \texttt{[P]} \end{array}$$

`dip`:

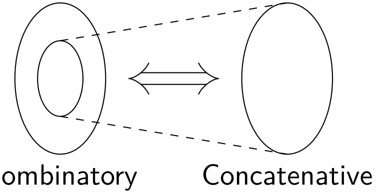
$$\begin{array}{rcl} \underline{C\,B}\;q\;\alpha\;x & \Leftrightarrow & x\;\texttt{[P] dip} \\ B\;\alpha\;q\;x & & \\ \alpha\,(q\;x) & \Leftrightarrow & \texttt{P}\;x \end{array}$$

$C\,I\alpha$ is regular.
$C\,B$ is **regular-ish**: if α is regular, these are too!

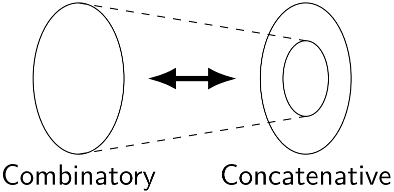
From Combinatory to Concatenative

The simulation $\Leftrightarrow$ only relates regular-ish combinators.
We want to relate any combinator with a concatenative program.

What we have:



We also want:



Call-by-Name vs Call-by-Value

Kerby (2002) provided a simulation which relates any combinator.

His simulation simulates call-by-name execution of combinators.

call-by-name:

$$\begin{array}{r} B\,(B\,C)\;K\;x\;y\;z\;w \\ B\;C\,(K\,x)\;y\;z\;w \\ C\,(K\,x\;y)\;z\;w \\ \underline{K\;x\;y\;w\;z} \\ x\;w\;z \end{array}$$

call-by-value:

$$\begin{array}{r} B\,(B\,C)\;K\;x\;y\;z\;w \\ B\;C\,(K\,x)\;y\;z\;w \\ C\,(K\,x\;y)\;z\;w \\ \underline{C\;x\;z\;w} \\ x\;w\;z \end{array}$$

We provide two simulations for call-by-value.

Our Contributions (Thanks!)

We contribute with four simulations:

Combinatory		Concatenative
Regular	$\Longleftrightarrow$	First-Order
Regular-ish	$\Longleftrightarrow$	Higher-Order
Untyped	$\longleftrightarrow$	Dynamic call-by-value
Simply-typed	$\longleftrightarrow$	Call-by-value is possible

How to find me:

0. `dkhashimoto@ic.ufrj.br`
1. github.com/Kiyoshi364/static-memory

INSTITUTO DE COMPUTAÇÃO UFRJ

Values:

$$f \sim f \qquad \frac{\alpha \leftrightarrow \texttt{P}}{\alpha \sim [\ \texttt{P}\ ]}$$

First-order:

$$B \leftrightarrow \texttt{apply} \qquad C \leftrightarrow \texttt{swap} \qquad K \leftrightarrow \texttt{zap} \qquad W \leftrightarrow \texttt{dup} \qquad I \leftrightarrow \varepsilon$$

$$\frac{\alpha \leftrightarrow \texttt{P} \qquad \beta \leftrightarrow \texttt{Q}}{B\,\alpha\,\beta \leftrightarrow \texttt{P Q}} \qquad \frac{\alpha \leftrightarrow \texttt{P}}{B\,\alpha \leftrightarrow [\ \texttt{P}\ ]\ \texttt{dip}} \qquad \frac{\alpha \sim x \qquad \beta \leftrightarrow \texttt{Q}}{C\,\beta\,\alpha \leftrightarrow x\ \texttt{Q}}$$

Higher-order:

$$C\,I \leftrightarrow \texttt{call} \qquad C\,B \leftrightarrow \texttt{dip} \qquad C\,(B\,B\,B)\,C \leftrightarrow \texttt{cons}$$

Simulation:

$$\frac{\alpha \leftrightarrow \texttt{P}}{\alpha\ q \Leftrightarrow \texttt{P}} \qquad q \Leftrightarrow \varepsilon \qquad \frac{\alpha \sim x}{q\ \alpha \Leftrightarrow x} \qquad \frac{\hat{\alpha} \Leftrightarrow \texttt{P} \qquad \hat{\beta} \Leftrightarrow \texttt{Q}}{\hat{\alpha}\{\hat{\beta}/q\} \Leftrightarrow \texttt{P Q}}$$

Regular-ish and Higher-Order (Pushing, popping and partial application)

$C\, I \Leftrightarrow$ `call`:

$$\begin{aligned} C\, I\, q\, \alpha \;&\Leftrightarrow\; \texttt{[P] call} \\ I\, \alpha\, q & \\ \alpha\, q \;&\Leftrightarrow\; \texttt{P} \end{aligned}$$

$C\, B \Leftrightarrow$ `dip`:

$$\begin{aligned} C\, B\, q\, \alpha\, \varphi \;&\Leftrightarrow\; x\ \texttt{[P] dip} \\ B\, \alpha\, q\, \varphi & \\ \alpha\, (q\, \varphi) \;&\Leftrightarrow\; \texttt{P}\ x \end{aligned}$$

$C\, (B\, B\, B)\, C \Leftrightarrow$ `cons`:

$$\begin{aligned} C\, (B\, B\, B)\, C\, q\, \alpha\, \varphi \;&\Leftrightarrow\; x\ \texttt{[P] cons} \\ B\, B\, B\, q\, C\, \alpha\, \varphi & \\ B\, (B\, q)\, C\, \alpha\, \varphi & \\ B\, q\, (C\, \alpha)\, \varphi & \\ q\, (C\, \alpha\, \varphi) \;&\Leftrightarrow\; \texttt{[}\ x\ \texttt{P]} \end{aligned}$$

$C\, \alpha\, \varphi \Leftrightarrow \texttt{[}\ x\ \texttt{P]}$:

$$\begin{aligned} C\, \alpha\, \varphi\, q \;&\Leftrightarrow\; x\ \texttt{P} \\ \alpha\, q\, \varphi \;&\Leftrightarrow\; x\ \texttt{P} \end{aligned}$$

$\alpha \leftrightarrow$ P as a Function (From Concatenative to Combinatory)

$$
\begin{aligned}
[\![\texttt{apply}]\!] &= B \\
[\![\texttt{swap}]\!] &= C \\
[\![\texttt{zap}]\!] &= K \\
[\![\texttt{dup}]\!] &= W \\
[\![\varepsilon]\!] &= I \\
[\![x\ \texttt{P}]\!] &= C\,[\![\texttt{P}]\!]\,x
\end{aligned}
\qquad
\begin{aligned}
[\![\texttt{P Q}]\!] &= B\,[\![\texttt{P}]\!]\,[\![\texttt{Q}]\!] \\
[\![\texttt{[P] dip}]\!] &= B\,[\![\texttt{P}]\!] \\
[\![\texttt{dip}]\!] &= C\,B \\
[\![\texttt{call}]\!] &= C\,I \\
[\![\texttt{cons}]\!] &= C\,(B\,B\,B)\,C
\end{aligned}
$$

From Combinatory to Concatenative

Kerby (2002) provided a simple simulation:

$$
\begin{array}{lcllcr}
«\,B\,» & := & [\,\texttt{cons}\,]\ \texttt{dip call} & \qquad «W» & := & [\,\texttt{dup}\,]\ \texttt{dip call} \\
«\,C\,» & := & [\,\texttt{swap}\,]\ \texttt{dip call} & \qquad «\,I\,» & := & \texttt{call} \\
«\,K\,» & := & [\,\texttt{zap}\,]\ \texttt{dip call} & \qquad «\alpha\ \beta» & := & [\,«\beta»\,]\ «\alpha»
\end{array}
$$

- combinators match same instructions
- `dip` skips over the continuation
- `call` executes the continuation

This simulation works well for call-by-name, but not for call-by-value.
We provide two working simulations for call-by-value.

Total and Partial Application (From Combinatory to Concatenative)

Concatenative has different primitives for total and partial application.

In call-by-value, red B inserts `call` in the middle of the program:

$$
\begin{aligned}
B\,K\,I\,x\,y \iff{}& y\;x\;[\,\dot{I}\,]\;[\,\dot{K}\,]\;[\,\texttt{call}\,]\;\texttt{dip}\;\texttt{call}\\
& y\;x\;[\,\dot{I}\,]\;\texttt{call}\;[\,\dot{K}\,]\;\texttt{call}\\
K\,(I\,x)\,y \iff{}& y\;x\;\dot{I}\;[\,\dot{K}\,]\;\texttt{call}
\end{aligned}
$$

Blue B inserts `cons`:

$$
\begin{aligned}
B\,I\,K\,x\,y \iff{}& y\;x\;[\,\dot{K}\,]\;[\,\dot{I}\,]\;[\,\texttt{cons}\,]\;\texttt{dip}\;\texttt{call}\;\texttt{call}\\
& y\;x\;[\,\dot{K}\,]\;\texttt{cons}\;[\,\dot{I}\,]\;\texttt{call}\;\texttt{call}\\
I\,(K\,x)\,y \iff{}& y\;[\,x\;\dot{K}\,]\;[\,\dot{I}\,]\;\texttt{call}\;\texttt{call}
\end{aligned}
$$

How to know if the B is red or blue:

0. dynamic choice: quotations count remaining arguments at runtime
1. static choice: simply-typed combinators

Dynamic Concatenative (From Combinatory to Concatenative)

The dynamic instruction $\star$:

$$x\ [\,\mathtt{P}\,]_n \star \ \mapsto\ x\,\mathtt{P} \qquad \text{if } n = 1 \text{ and } x \text{ is a value}$$
$$x\ [\,\mathtt{P}\,]_n \star \ \mapsto\ [\,x\,\mathtt{P}\,]_{n-1} \qquad \text{if } n \geq 2 \text{ and } x \text{ is a value}$$

Dynamic call-by-value simulation:

$$\begin{aligned} \langle\, B \,\rangle &:= [\,[\star]\,\mathtt{dip}\star\,]_3 & \langle\, K \,\rangle &:= [\,[\mathtt{zap}]\,\mathtt{dip}\,]_2 \\ \langle\, C \,\rangle &:= [\,[\mathtt{swap}]\,\mathtt{dip}\star\star\,]_3 & \langle\, I \,\rangle &:= []_1 \\ \langle W \rangle &:= [\,[\,\mathtt{dup}\,]\,\mathtt{dip}\star\star\,]_2 & \langle \alpha\ \beta \rangle &:= \langle\beta\rangle\ \langle\alpha\rangle\ \star \end{aligned}$$

Inferring cons and call (From Combinatory to Concatenative)

$$B : (b \xrightarrow{x} c) \xrightarrow{\texttt{cons}} (b \xrightarrow{y} a) \xrightarrow{\texttt{cons}} a \xrightarrow{\texttt{call}} c \Rightarrow \texttt{[[y] dip x]}$$

$$C : (a \xrightarrow{x} b \xrightarrow{y} c) \xrightarrow{\texttt{cons}} b \xrightarrow{\texttt{cons}} a \xrightarrow{\texttt{call}} c \Rightarrow \texttt{[[swap] dip x y]}$$

$$W : (a \xrightarrow{x} a \xrightarrow{y} b) \xrightarrow{\texttt{cons}} a \xrightarrow{\texttt{call}} b \Rightarrow \texttt{[[dup] dip x y]}$$

$$K : a \xrightarrow{\texttt{cons}} b \xrightarrow{\texttt{call}} a \Rightarrow \texttt{[[zap] dip]} \qquad I : a \xrightarrow{\texttt{call}} a \Rightarrow \texttt{[ ]}$$

$$\frac{\alpha : a \xrightarrow{x} b \Rightarrow \texttt{P} \qquad \beta : a \Rightarrow \texttt{Q}}{\alpha\ \beta : b \Rightarrow \texttt{Q P x}}$$

Inferring example (From Combinatory to Concatenative)

$B\ K\ I\ x\ y \Rightarrow y\ x$ `[ ] [ [zap] dip ] [ [call`$_{(1)}$ `] dip cons`$_{(2)}$ `]` `cons`$_{(3)}$ `cons`$_{(4)}$ `call`$_{(5)}$ `call`$_{(6)}$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c} B : (a \overset{\texttt{cons}\,\mathbf{(2)}}{\longrightarrow} (b \overset{\texttt{call}\,6}{\longrightarrow} a)) \overset{\texttt{cons}\,\mathbf{(3)}}{\longrightarrow} (a \overset{\texttt{call}\,\mathbf{(1)}}{\longrightarrow} a) \overset{\texttt{cons}\,4}{\longrightarrow} a \overset{\texttt{call}\,5}{\longrightarrow} (b \overset{\texttt{call}\,6}{\longrightarrow} a) \\ K : a \overset{\texttt{cons}\,2}{\longrightarrow} b \overset{\texttt{call}\,6}{\longrightarrow} a \end{array}}{\begin{array}{c} B\ K : (a \overset{\texttt{call}\,1}{\longrightarrow} a) \overset{\texttt{cons}\,\mathbf{(4)}}{\longrightarrow} a \overset{\texttt{call}\,5}{\longrightarrow} (b \overset{\texttt{call}\,6}{\longrightarrow} a) \\ I : a \overset{\texttt{call}\,1}{\longrightarrow} a \end{array}}}{B\ K\ I : a \overset{\texttt{call}\,\mathbf{(5)}}{\longrightarrow} (b \overset{\texttt{call}\,6}{\longrightarrow} a) \qquad\qquad x : a}}{\begin{array}{c} B\ K\ I\ x : b \overset{\texttt{call}\,\mathbf{(6)}}{\longrightarrow} a \\ y : b \end{array}}}{B\ K\ I\ x\ y : a}
$$

Dummy end