Connections Between Applicative and Concatenative Tacit Programming

Hashimoto, Daniel Kiyoshi

November 2025 – TalTech, Tallinn, Estonia

About me

I am finishing my Master's Degree

- Universidade Federal do Rio de Janeiro, Brazil
- Bachelor's in Computer Science
- I study Programming Languages

About me

I am finishing my Master's Degree

- Universidade Federal do Rio de Janeiro, Brazil
- Bachelor's in Computer Science
- I study Programming Languages

I'm looking for a PhD program

- concatenative languages are similar to string diagrams
- string diagrams for first-order logic

Where I keep my publications and programming projects

- `github.com/Kiyoshi364/static-memory`

My Research Interests

Interesting research topics (probably non-exhaustive):

- alternative programming paradigms and alternative computing models
- theorem proving and proof assistants
- static analysis, type systems, logic systems
- creating and using models

My Research Interests

Interesting research topics (probably non-exhaustive):

- alternative programming paradigms and alternative computing models
- theorem proving and proof assistants
- static analysis, type systems, logic systems
- creating and using models

I like research that *roughly* follows these steps:

1. point at two things;
2. provide a definition of “equality”;
3. claim: “those two things are equal”

My Research Interests

Interesting research topics (probably non-exhaustive):

- alternative programming paradigms and alternative computing models
- theorem proving and proof assistants
- static analysis, type systems, logic systems
- creating and using models

I like research that *roughly* follows these steps:

1. point at two things;
2. provide a definition of “equality”;
3. claim: “those two things are equal”

(Often, the research starts in a 1-3-2 order)

Connections Between Applicative and Concatenative Tacit Programming

Hashimoto, Daniel Kiyoshi

November 2025 – TalTech, Tallinn, Estonia

Programming without named variables
Succinct programs with emphasis on data flow

We highlight two styles

1.

2.

Tacit (Point-free) Programming

Programming without named variables
Succinct programs with emphasis on data flow

We highlight two styles

1. Applicative Style (higher-order functions)

$$h := \lambda x \,.\, f\,(g\,x) \qquad\qquad h := f \circ g \tag{3}$$

2.

Tacit (Point-free) Programming

Programming without named variables
Succinct programs with emphasis on data flow

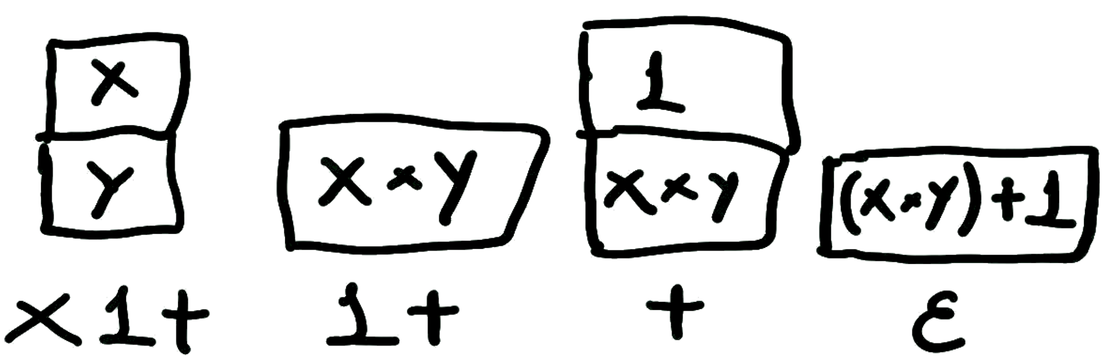
We highlight two styles

1. Applicative Style (higher-order functions)

$$h := \lambda x\,.\,f\;(g\;x) \qquad\qquad h := f \circ g \tag{5}$$

2. Compositive Style (stack-based programming)

$$h := \lambda x\;y\,.(x \times y) + 1 \qquad\qquad h := \times\;1\;+ \tag{6}$$



Tacit (Point-free) Programming

Programming without named variables
Succinct programs with emphasis on data flow

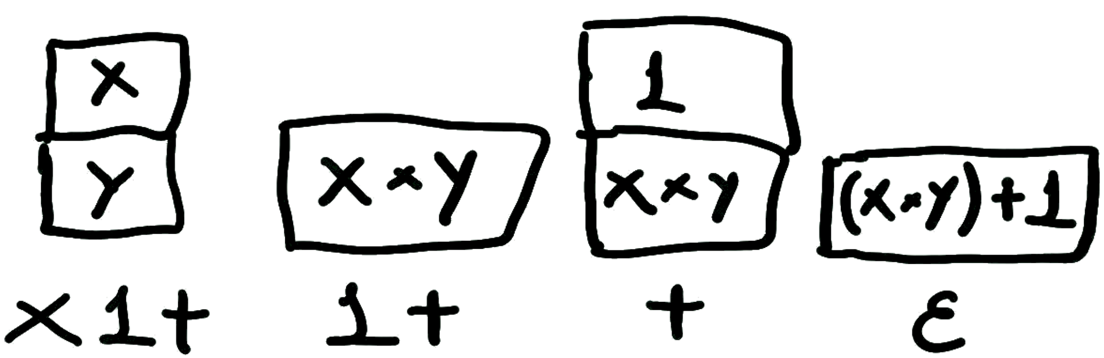
We highlight two styles

1. Applicative Style (higher-order functions)

$$h := \lambda x \,.\, f\,(g\,x) \qquad\qquad h := f \circ g \tag{7}$$

2. Compositive Style (stack-based programming)

$$h := \lambda x\,y\,.(x \times y) + 1 \qquad\qquad h := \times\; 1\; + \tag{8}$$



In both cases, there were no x nor y

Intuition for “Equality”

Example for BCK and $\mathsf{swap}\ \mathsf{zap}$

$$B\,C\,K\,q\,x\,y \qquad \trianglerighteq \qquad y\,x \mid \mathsf{swap}\ \mathsf{zap} \tag{9}$$

Example for BCK and $\mathsf{swap}\;\mathsf{zap}$

$$\begin{aligned} B\,C\,K\,q\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\;\mathsf{zap} \\ C\,(K\,q)\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\;\mathsf{zap} \end{aligned} \tag{10}$$

Intuition for "Equality"

Example for BCK and `swap zap`

$$\begin{aligned} B\,C\,K\,q\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\;\mathsf{zap} \\ C\,(K\,q)\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\;\mathsf{zap} \\ K\,q\,y\,x &\quad\trianglerighteq\quad x\,y \mid \mathsf{zap} \end{aligned} \tag{11}$$

Intuition for "Equality"

Example for BCK and `swap zap`

$$\begin{aligned} B\,C\,K\,q\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\ \mathsf{zap} \\ C\,(K\,q)\,x\,y &\quad\trianglerighteq\quad y\,x \mid \mathsf{swap}\ \mathsf{zap} \\ K\,q\,y\,x &\quad\trianglerighteq\quad x\,y \mid \mathsf{zap} \\ q\,x &\quad\trianglerighteq\quad x \mid \varepsilon \end{aligned} \tag{12}$$

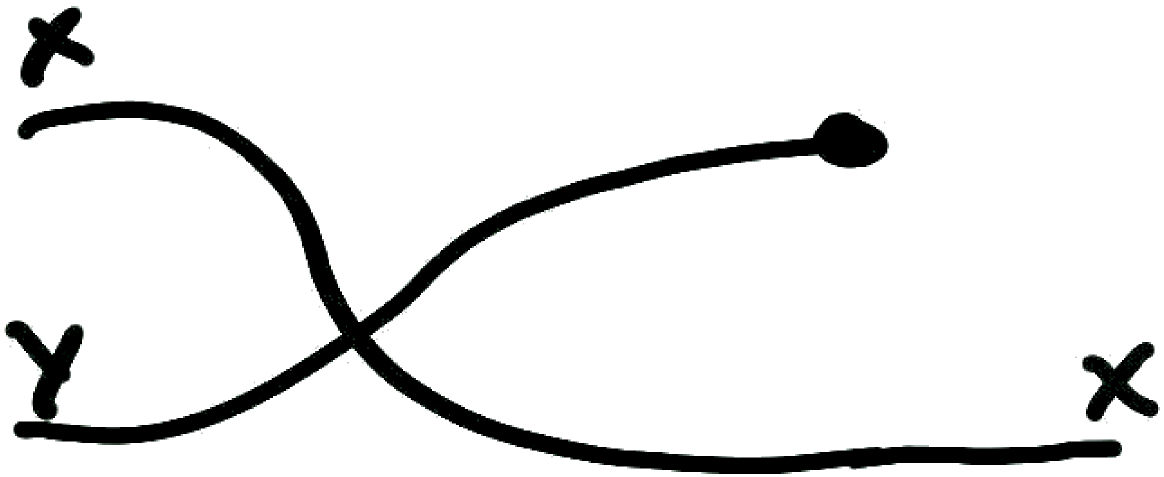
The programs start related
If I step on one size, the other side is also related after some steps
(This is called (Bi)Simulation, I'll introduce it latter)

Intuition for “Equality”

Example for BCK and `swap zap`

$$\begin{array}{rcl} B\,C\,K\,q\,x\,y & \triangleright & y\,x \mid \texttt{swap zap} \\ C\,(K\,q)\,x\,y & \triangleright & y\,x \mid \texttt{swap zap} \\ K\,q\,y\,x & \triangleright & x\,y \mid \texttt{zap} \\ q\,x & \triangleright & x \mid \varepsilon \end{array} \tag{13}$$

The programs start related
If I step on one size, the other side is also related after some steps
(This is called (Bi)Simulation, I’ll introduce it latter)

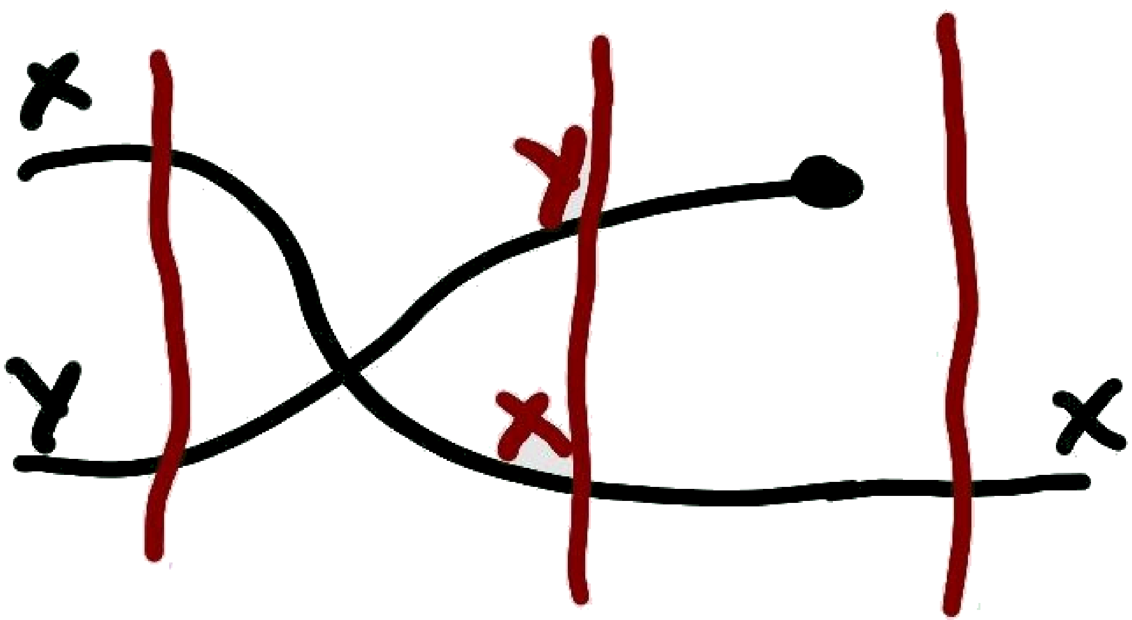


Intuition for "Equality"

Example for BCK and $\mathtt{swap}\ \mathtt{zap}$

$$\begin{array}{rcr} B\,C\,K\,q\,x\,y & \triangleright & y\,x \mid \mathtt{swap}\ \mathtt{zap} \\ C\,(K\,q)\,x\,y & \triangleright & y\,x \mid \mathtt{swap}\ \mathtt{zap} \\ K\,q\,y\,x & \triangleright & x\,y \mid \mathtt{zap} \\ q\,x & \triangleright & x \mid \varepsilon \end{array} \tag{14}$$

The programs start related
If I step on one size, the other side is also related after some steps
(This is called (Bi)Simulation, I'll introduce it latter)



A model for higher-order stack languages Thun (1994), Kerby (2002), Kleffner (2017)

$\mathrm{Cat} = \mathrm{list}(\mathrm{Instr} + \mathrm{Value})$

A model for higher-order stack languages Thun (1994), Kerby (2002), Kleffner (2017)
$\text{Cat} = \text{list}(\text{Instr} + \text{Value})$

Each instruction solves a problem

$$\begin{array}{ll} \text{Effect} & \text{Instructions} \\ \text{Permute} & y\,x\,\texttt{swap} \mapsto x\,y \\ \text{Discard} & x\,\texttt{zap} \mapsto \varepsilon \\ \text{Duplicate} & x\,\texttt{dup} \mapsto x\,x \\ \text{Dequote} & \texttt{[}\rho\texttt{]}\,\texttt{call} \mapsto \rho \\ \text{Quotation Creation} & x\,\texttt{unit} \mapsto \texttt{[}x\texttt{]} \\ \text{Quotation Composition} & \texttt{[}\pi\texttt{]}\,\texttt{[}\rho\texttt{]}\,\texttt{cat} \mapsto \texttt{[}\,\pi\,\rho\,\texttt{]} \\ \text{Stash} & x\,\texttt{[}\rho\texttt{]}\,\texttt{dip} \mapsto \rho\,x \end{array} \tag{17}$$

Recursive/Structural rule

$$\frac{\sigma, \pi : \text{Cat} \qquad \rho_0 \mapsto \rho}{\sigma\,\rho_0\,\pi \mapsto \sigma\,\rho\,\pi} \tag{18}$$

Quotations and Higher-Order Instructions

A quotation is an anonymous function (lambda)
We can put quotations on the stack

Higher-order instructions use and/or create quotations

`call` executes a quotation

$$x\ \texttt{[dup]}\ \texttt{call} \mapsto x\ \texttt{dup} \mapsto x\ x \tag{19}$$

Quotations and Higher-Order Instructions

A quotation is an anonymous function (lambda)
We can put quotations on the stack

Higher-order instructions use and/or create quotations

`call` executes a quotation

$$x\ \texttt{[dup]}\ \texttt{call} \mapsto x\ \texttt{dup} \mapsto x\ x \tag{22}$$

`unit` creates a quotation

$$x\ \texttt{unit} \mapsto \texttt{[}x\texttt{]} \tag{23}$$

A quotation is an anonymous function (lambda)
We can put quotations on the stack

Higher-order instructions use and/or create quotations

`call` executes a quotation

$$x\ \texttt{[dup]}\ \texttt{call} \mapsto x\ \texttt{dup} \mapsto x\ x \tag{25}$$

`unit` creates a quotation

$$x\ \texttt{unit} \mapsto \texttt{[}x\texttt{]} \tag{26}$$

`cat` concatenates/composes quotations

$$\texttt{[zap]}\ \texttt{[}\rho\texttt{]}\ \texttt{cat} \mapsto \texttt{[zap}\ \rho\ \texttt{]} \tag{27}$$

> Syntactical concatenation is semantical composition!

Stack Rotations vs The dip Instruction

The standard way is to use rotations for stack shuffling

$$z\;y\;x\;\texttt{dig3}\;\texttt{swap} \mapsto y\;x\;z\;\texttt{swap} \mapsto y\;z\;x \tag{28}$$

The standard way is to use rotations for stack shuffling

$$z\;y\;x\;\texttt{dig3}\;\texttt{swap} \mapsto y\;x\;z\;\texttt{swap} \mapsto y\;z\;x \tag{30}$$

Instead, we use `dip`

`dip` runs a program one level deeper

$$z\;y\;x\;\texttt{[swap]}\;\texttt{dip} \mapsto z\;y\;\texttt{swap}\;x \mapsto y\;z\;x \tag{31}$$

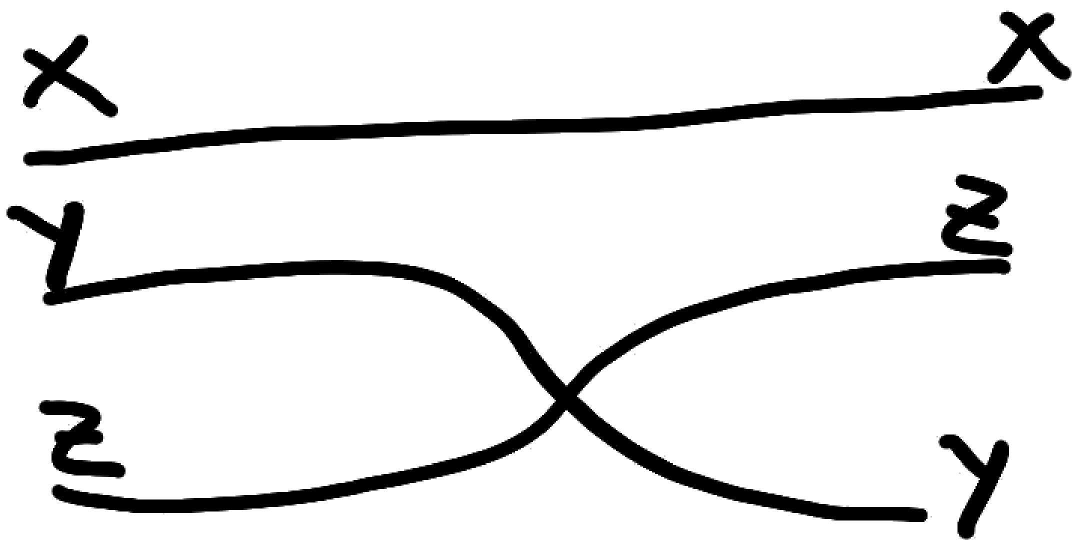
The standard way is to use rotations for stack shuffling

$$z\;y\;x\;\texttt{dig3}\;\texttt{swap} \mapsto y\;x\;z\;\texttt{swap} \mapsto y\;z\;x \tag{32}$$

Instead, we use `dip`
`dip` runs a program one level deeper

$$z\;y\;x\;\texttt{[swap]}\;\texttt{dip} \mapsto z\;y\;\texttt{swap}\;x \mapsto y\;z\;x \tag{33}$$

In the diagrams, `dip` is equivalent to putting an identity on top of the program



First-Order Concatenative Calculus

A model for a stack languages which cannot put functions on the stack

$$y\, x\, \texttt{swap} \mapsto x\, y \qquad\qquad x\, \texttt{zap} \mapsto \varepsilon$$

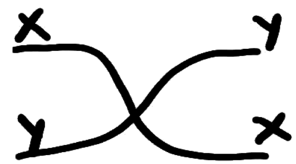
$$x\, \texttt{dup} \mapsto x\, x \qquad\qquad \pi\, \rho$$

$$x\, (\texttt{[}\rho\texttt{]}\, \texttt{dip}) \mapsto \rho\, x \qquad\qquad \varepsilon$$

First-Order Concatenative Calculus and String Diagrams

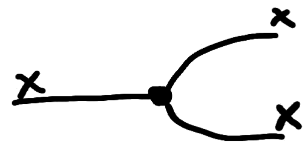
A model for a stack languages which cannot put functions on the stack

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

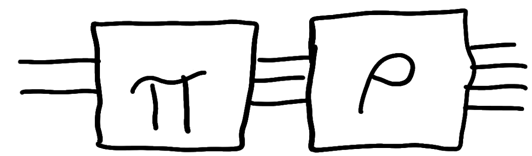


$x\; \texttt{zap} \;\mapsto\; \varepsilon$

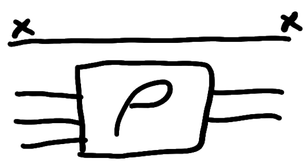
$x\; \texttt{dup} \;\mapsto\; x\; x$



$\pi\; \rho$



$x\; (\texttt{[}\rho\texttt{]}\; \texttt{dip}) \;\mapsto\; \rho\; x$

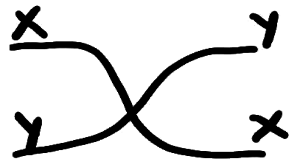


ε

Given a first-order program, I can draw a string diagram
(related to Cartesian Monoidal Categories?)

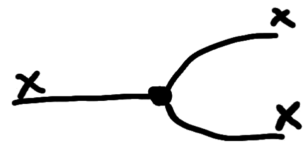
A model for a stack languages which cannot put functions on the stack

$y\,x\,\texttt{swap} \;\mapsto\; x\,y$

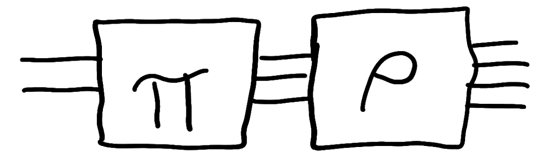


$x\,\texttt{zap} \;\mapsto\; \varepsilon$

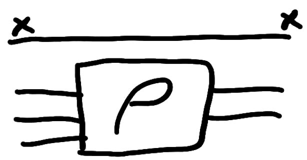
$x\,\texttt{dup} \;\mapsto\; x\,x$



$\pi\,\rho$



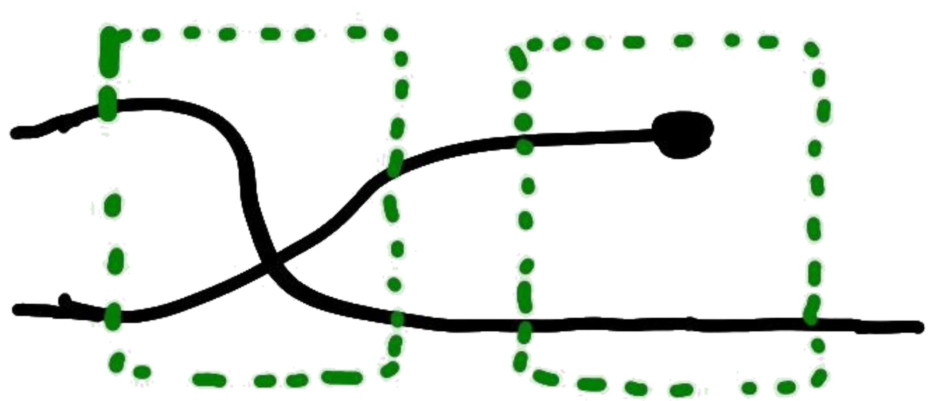
$x\,(\texttt{[}\rho\texttt{]}\,\texttt{dip}) \;\mapsto\; \rho\,x$



ε

Given a first-order program, I can draw a string diagram
(related to Cartesian Monoidal Categories?)

If the arities don't match, one can put extra identities on the bottom



How to Draw (Higher-Order) Concatenative Calculus?

vvv This slide is from before I arrived here vvv

I don't know how to draw an arbitrary Concatenative Program

A quotation (which is in a wire) should become a diagram

For instance:

$$\texttt{swap}\,\texttt{call} \tag{34}$$

Any ideas? (Perhaps, after more higher-order examples)

^^^ This slide is from before I arrived here ^^^

(related to Cartesian Closed Monoidal Category?)

Model for applicative functional languages
"lambda calculus without variables"

Each combinator does an effect

$$
\begin{array}{lcl}
\text{Effect} & & \text{Combinators} \\
\\
\text{Permute} & \mathrm{C} & q\,x\,y \longrightarrow q\,y\,x \\
\text{Discard} & \mathrm{K} & q\,x \longrightarrow q \\
\text{Duplicate} & \mathrm{W} & q\,x \longrightarrow q\,x\,x \\
\text{Identity} & \mathrm{I} & q \longrightarrow q \\
\text{Composition*} & \mathrm{B} & q\,x\,y \longrightarrow q\,(x\,y)
\end{array}
\tag{35}
$$

Created by Schönfinkel (1924), expanded and popularized by Curry (1930)

Model for applicative functional languages
"lambda calculus without variables"

Each combinator does an effect

$$
\begin{array}{ll}
\text{Effect} & \text{Combinators} \\
\text{Permute} & q\,x\,y \longrightarrow q\,y\,x \\
\text{Discard} & q\,x \longrightarrow q \\
\text{Duplicate} & q\,x \longrightarrow q\,x\,x \\
\text{Identity} & q \longrightarrow q \\
\text{Composition}^* & q\,x\,y \longrightarrow q\,(x\,y)
\end{array}
\tag{36}
$$

Created by Schönfinkel (1924), expanded and popularized by Curry (1930)

Smullyan (1985) provided bird names for combinators

Model for applicative functional languages
"lambda calculus without variables"

Each combinator does an effect

$$\begin{array}{lrllll} \text{Effect} & & \text{Combinators} & & & \\ & & & & & \\ \text{Permute} & \mathrm{C} & q\,x\,y & \longrightarrow & q\,y\,x \\ \text{Discard} & \mathrm{K} & q\,x & \longrightarrow & q \\ \text{Duplicate} & \mathrm{W} & q\,x & \longrightarrow & q\,x\,x \\ \text{Identity} & \mathrm{I} & q & \longrightarrow & q \\ \text{Composition*} & \mathrm{B} & q\,x\,y & \longrightarrow & q\,(x\,y) \end{array} \tag{37}$$

Created by Schönfinkel (1924), expanded and popularized by Curry (1930)

Smullyan (1985) provided bird names for combinators

Regular Combinators

Regular Combinators keep the first argument q in place, without extra copies
q is a **continuation** (what is the program doing next)
It can reorganize the other arguments in any way

All basic combinators are regular

$$\begin{array}{llll} \mathrm{C}\; q\,x\,y \longrightarrow q\,y\,x & & \mathrm{K}\; q\,x \longrightarrow q \\ \mathrm{W}\; q\,x \longrightarrow q\,x\,x & & \mathrm{I}\; q \longrightarrow q \\ \mathrm{B}\; q\,x\,y \longrightarrow q\,(x\,y) & & \end{array} \tag{38}$$

Regular Combinators

Regular Combinators keep the first argument q in place, without extra copies
q is a **continuation** (what is the program doing next)
It can reorganize the other arguments in any way

All basic combinators are regular

$$\begin{array}{llll} \mathrm{C} & q\,x\,y & \longrightarrow & q\,y\,x \\ \mathrm{W} & q\,x & \longrightarrow & q\,x\,x \\ \mathrm{B} & q\,x\,y & \longrightarrow & q\,(x\,y) \end{array} \qquad \begin{array}{llll} \mathrm{K} & q\,x & \longrightarrow & q \\ \mathrm{I} & q & \longrightarrow & q \end{array} \tag{41}$$

BCK is regular

$$B\,C\,K\,q\,x\,y \longrightarrow C\,(K\,q)\,x\,y \longrightarrow K\,q\,y\,x \longrightarrow q\,x \tag{42}$$

Both, CI and WC, are irregular

$$\begin{gathered} C\,I\,q\,x \longrightarrow I\,x\,q \longrightarrow x\,q \\ W\,C\,q\,x \longrightarrow C\,q\,q\,x \longrightarrow q\,x\,q \end{gathered} \tag{43}$$

Regular Combinators by Construction

We can construct regular combinators with these rules:

$$
\begin{aligned}
\mathrm{C}\; q\; x\; y &\longrightarrow q\; y\; x \\
\mathrm{K}\; q\; x &\longrightarrow q \\
\mathrm{W}\; q\; x &\longrightarrow q\; x\; x \\
\mathrm{I}\; q &\longrightarrow q \\
\\
\mathrm{B}\; q\; x\; y &\longrightarrow q\; (x\; y) \\
(B\; \alpha)\; q\; x &\longrightarrow \alpha\; (q\; x) \\
(B\; \alpha\; \beta)\; q &\longrightarrow \alpha\; (\beta\; q) \\
\\
(C\; \alpha\; x)\; q &\longrightarrow \alpha\; q\; x
\end{aligned}
\tag{44}
$$

B also stashes arguments and sequences regular combinators!

C also introduces new values!

We can construct regular combinators with these rules:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\mathrm{C}\; q\, x\, y & \longrightarrow & q\, y\, x & y\, x\, \texttt{swap} & \mapsto & x\, y \\
\mathrm{K}\; q\, x & \longrightarrow & q & x\, \texttt{zap} & \mapsto & \varepsilon \\
\mathrm{W}\; q\, x & \longrightarrow & q\, x\, x & x\, \texttt{dup} & \mapsto & x\, x \\
\mathrm{I}\; q & \longrightarrow & q & & \varepsilon & \\
\\
\mathrm{B}\; q\, x\, y & \longrightarrow & q\,(x\, y) & & \times & \\
(B\, \alpha)\; q\, x & \longrightarrow & \alpha\,(q\, x) & x\,(\texttt{[}\rho\texttt{]}\; \texttt{dip}) & \mapsto & \rho\, x \\
(B\, \alpha\, \beta)\; q & \longrightarrow & \alpha\,(\beta\, q) & & \rho\, \pi & \\
\\
(C\, \alpha\, x)\; q & \longrightarrow & \alpha\, q\, x & & x\, \rho &
\end{array}
\tag{45}
$$

Besides B with 0 arguments,
regular combinators map to first-order concatenative

Shuffle combinators keep q in place, without extra copies
and can only shuffle the other arguments (cannot add parenthesis)

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\mathrm{C}\, q\, x\, y & \longrightarrow & q\, y\, x & y\, x\, \texttt{swap} & \mapsto & x\, y \\
\mathrm{K}\, q\, x & \longrightarrow & q & x\, \texttt{zap} & \mapsto & \varepsilon \\
\mathrm{W}\, q\, x & \longrightarrow & q\, x\, x & x\, \texttt{dup} & \mapsto & x\, x \\
\mathrm{I}\, q & \longrightarrow & q & & \varepsilon & \\
\\
(B\, \alpha)\, q\, x & \longrightarrow & \alpha\, (q\, x) & x\, (\texttt{[}\rho\texttt{]}\, \texttt{dip}) & \mapsto & \rho\, x \\
(B\, \alpha\, \beta)\, q & \longrightarrow & \alpha\, (\beta\, q) & & \rho\, \pi & \\
\\
(C\, \alpha\, x)\, q & \longrightarrow & \alpha\, q\, x & & x\, \rho &
\end{array}
\tag{46}
$$

Shuffle combinators keep q in place, without extra copies
and can only shuffle the other arguments (cannot add parenthesis)

$$\begin{array}{rcl}
\mathrm{C}\,q\,x\,y & \longrightarrow & q\,y\,x \\
\mathrm{K}\,q\,x & \longrightarrow & q \\
\mathrm{W}\,q\,x & \longrightarrow & q\,x\,x \\
\mathrm{I}\,q & \longrightarrow & q \\
\\
(B\,\alpha)\,q\,x & \longrightarrow & \alpha\,(q\,x) \\
(B\,\alpha\,\beta)\,q & \longrightarrow & \alpha\,(\beta\,q) \\
\\
(C\,\alpha\,x)\,q & \longrightarrow & \alpha\,q\,x
\end{array}
\qquad
\begin{array}{rcl}
y\,x\,\texttt{swap} & \mapsto & x\,y \\
x\,\texttt{zap} & \mapsto & \varepsilon \\
x\,\texttt{dup} & \mapsto & x\,x \\
 & \varepsilon & \\
\\
x\,(\texttt{[}\rho\texttt{]}\,\texttt{dip}) & \mapsto & \rho\,x \\
 & \rho\,\pi & \\
\\
 & x\,\rho &
\end{array}
\tag{47}$$

This title is imprecise! We need 2 relations:

1. $(\blacktriangleright)$: ShuffleComb $\times$ FOCat
2. $(\triangleright)$: ShuffleComb&q $\times$ FOCat

1. $(\blacktriangleright) : \text{ShuffleComb} \times \text{FOCat}$

$$C \blacktriangleright \texttt{swap} \qquad K \blacktriangleright \texttt{zap} \qquad W \blacktriangleright \texttt{dup} \qquad \overset{\text{R-EMPTY}}{I \blacktriangleright \varepsilon}$$

$$\frac{\alpha \blacktriangleright \rho}{B\,\alpha \blacktriangleright \texttt{[}\rho\texttt{]}\ \texttt{dip}} \qquad \overset{\text{R-CONCAT}}{\frac{\alpha \blacktriangleright \rho \quad \beta \blacktriangleright \pi}{B\,\alpha\,\beta \blacktriangleright \rho\,\pi}} \qquad \overset{\text{R-PUSH}}{\frac{\alpha \blacktriangleright \rho \quad x_0 \sim x_1}{C\,\alpha\,x_0 \blacktriangleright x_1\,\rho}} \tag{48}$$

2. $(\mathrel{\rhd\!\!\rhd}) : \text{ShuffleComb}\&q \times \text{FOCat}$

1. $(\blacktriangleright)$: ShuffleComb $\times$ FOCat

$$C \blacktriangleright \texttt{swap} \qquad K \blacktriangleright \texttt{zap} \qquad W \blacktriangleright \texttt{dup} \qquad \overset{\text{R-EMPTY}}{I \blacktriangleright \varepsilon}$$

$$\frac{\alpha \blacktriangleright \rho}{B\,\alpha \blacktriangleright \texttt{[}\rho\texttt{]}\ \texttt{dip}} \qquad \overset{\text{R-CONCAT}}{\frac{\alpha \blacktriangleright \rho \quad \beta \blacktriangleright \pi}{B\,\alpha\,\beta \blacktriangleright \rho\,\pi}} \qquad \overset{\text{R-PUSH}}{\frac{\alpha \blacktriangleright \rho \quad x_0 \sim x_1}{C\,\alpha\,x_0 \blacktriangleright x_1\,\rho}} \tag{50}$$

2. $(\triangleright)$: ShuffleComb&q $\times$ FOCat

$$\frac{\alpha \blacktriangleright \rho}{\alpha\,q \triangleright \rho}$$

$$\overset{\text{S-EMPTY}}{q \triangleright \varepsilon} \qquad \overset{\text{S-CONCAT}}{\frac{\hat{\alpha} \triangleright \rho \quad \hat{\beta} \triangleright \pi}{\hat{\alpha}\left\{\hat{\beta}/q\right\} \triangleright \rho\,\pi}} \qquad \overset{\text{S-PUSH}}{\frac{\hat{\alpha} \triangleright \rho \quad x_0 \sim x_1}{\hat{\alpha}\,x_0 \triangleright x_1\,\rho}} \tag{51}$$

- $\hat{\alpha}$ means a combinator with q inside
- $\hat{\alpha}\left\{\hat{\beta}/q\right\}$ substitutes $\hat{\beta}$ for q in $\hat{\alpha}$

Simulation Definition

A relation $(\trianglerighteq)$: $\mathrm{ShuffleComb}\&q \times \mathrm{FOCat}$ is a simulation for relations $(\longrightarrow)$: $\mathrm{ShuffleComb}\&q \times \mathrm{ShuffleComb}\&q$ and $(\mapsto)$: $\mathrm{FOCat} \times \mathrm{FOCat}$ iff

$$\begin{gathered}\forall(\hat{\alpha} : \mathrm{ShuffleComb}\&q)(\rho, \pi : \mathrm{FOCat}),\\ (\hat{\alpha} \trianglerighteq \rho) \wedge (\rho \mapsto^{*} \pi) \Rightarrow \exists\left(\hat{\beta} : \mathrm{ShuffleComb}\&q\right), \left(\hat{\alpha} \longrightarrow^{*} \hat{\beta}\right) \wedge \left(\hat{\beta} \trianglerighteq \pi\right)\end{gathered} \tag{52}$$

A relation $(\vartriangleright) : \text{ShuffleComb}\&q \times \text{FOCat}$ is a simulation for relations $(\longrightarrow) : \text{ShuffleComb}\&q \times \text{ShuffleComb}\&q$ and $(\mapsto) : \text{FOCat} \times \text{FOCat}$ iff

$$\begin{gathered} \forall(\hat{\alpha} : \text{ShuffleComb}\&q)(\rho, \pi : \text{FOCat}), \\ (\hat{\alpha} \vartriangleright \rho) \wedge (\rho \mapsto^{*} \pi) \Rightarrow \exists\left(\hat{\beta} : \text{ShuffleComb}\&q\right), \left(\hat{\alpha} \longrightarrow^{*} \hat{\beta}\right) \wedge \left(\hat{\beta} \vartriangleright \pi\right) \end{gathered} \tag{55}$$

or, with relational composition

$$(\vartriangleright) \ ; \ (\mapsto^{*}) \subseteq (\longrightarrow^{*}) \ ; \ (\vartriangleright) \tag{56}$$

or, with a diagram (unboxed are assumptions, boxed are results)

$$\begin{array}{ccc} \hat{\alpha} & \vartriangleright & \rho \\ \boxed{\downarrow_{*}} & & \mapsto^{*} \\ \boxed{\hat{\beta}} & \boxed{\vartriangleright} & \pi \end{array} \tag{57}$$

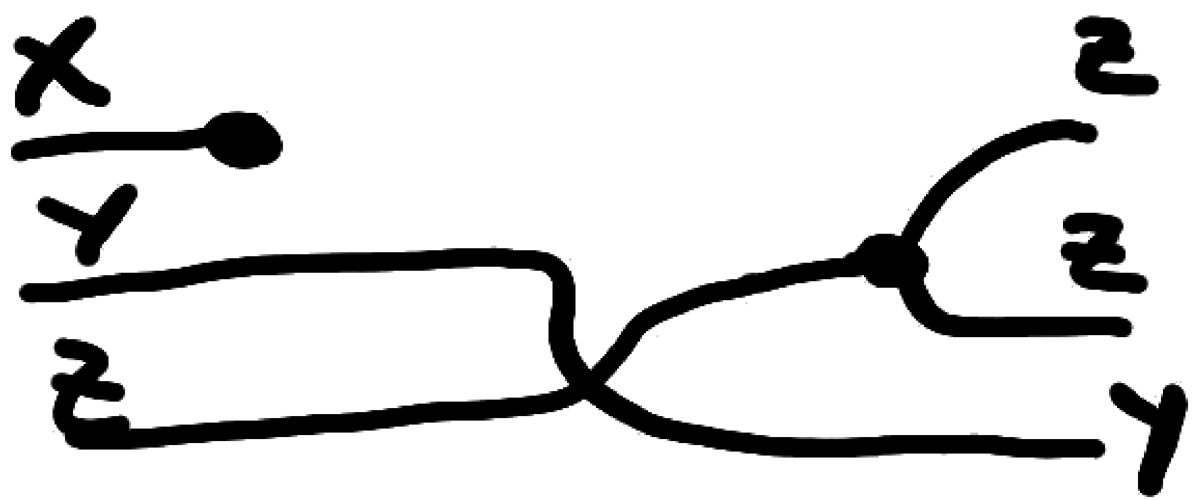
We may need some “bureaucratic steps” on the combinatory side

$$
\begin{array}{rcl}
B\,(B\,K\,C)\,W\,q\,x\,y\,z & \trianglerighteq & z\,y\,x\;\mathsf{zap}\;\mathsf{swap}\;\mathsf{dup} \\
B\,K\,C\,(W\,q)\,x\,y\,z & & \\
K\,(C\,(W\,q))\,x\,y\,z & & \\
C\,(W\,q)\,y\,z & \trianglerighteq & z\,y\;\mathsf{swap}\;\mathsf{dup} \\
W\,q\,z\,y & \trianglerighteq & y\,z\;\mathsf{dup} \\
q\,z\,z\,y & \trianglerighteq & y\,z\,z
\end{array}
\tag{58}
$$

Simulation Example

We may need some “bureaucratic steps” on the combinatory side

$$\begin{array}{rcl} B\,(B\,K\,C)\,W\,q\,x\,y\,z & \triangleright & z\;y\;x\;\mathsf{zap}\;\mathsf{swap}\;\mathsf{dup} \\ B\,K\,C\,(W\,q)\,x\,y\,z & & \\ K\,(C\,(W\,q))\,x\,y\,z & & \\ C\,(W\,q)\,y\,z & \triangleright & z\;y\;\mathsf{swap}\;\mathsf{dup} \\ W\,q\,z\,y & \triangleright & y\;z\;\mathsf{dup} \\ q\,z\,z\,y & \triangleright & y\;z\;z \end{array} \tag{59}$$



Simulation Definition (Again)

A relation $(\vartriangleright) : \text{ShuffleComb\&}q \times \text{FOCat}$ is a simulation for relations $(\longrightarrow) : \text{ShuffleComb\&}q \times \text{ShuffleComb\&}q$ and $(\mapsto) : \text{FOCat} \times \text{FOCat}$ iff

$$
\begin{gathered}
\forall(\hat{\alpha} : \text{ShuffleComb\&}q)(\rho, \pi : \text{FOCat}), \\
(\hat{\alpha} \vartriangleright \rho) \wedge (\rho \mapsto^* \pi) \Rightarrow \exists\left(\hat{\beta} : \text{ShuffleComb\&}q\right), \left(\hat{\alpha} \longrightarrow^* \hat{\beta}\right) \wedge \left(\hat{\beta} \vartriangleright \pi\right)
\end{gathered}
\tag{60}
$$

or, with relational composition

$$
(\vartriangleright) \; ; \; (\mapsto^*) \subseteq (\longrightarrow^*) \; ; \; (\vartriangleright) \tag{61}
$$

or, with a diagram (unboxed are assumptions, boxed are results)

$$
\begin{array}{ccc}
\hat{\alpha} & \vartriangleright & \rho \\
\boxed{\downarrow_*} & & \mathrel{\rotatebox[origin=c]{-90}{$\mapsto$}}_* \\
\boxed{\hat{\beta}} & \boxed{\vartriangleright} & \pi
\end{array}
\tag{62}
$$

“Co”simulation (Simulation of the Converse) Definition

A relation $(\rhd)$: $\mathsf{ShuffleComb}\&q \times \mathrm{FOCat}$ is a simulation for relations $(\longrightarrow)$: $\mathsf{ShuffleComb}\&q \times \mathsf{ShuffleComb}\&q$ and $(\mapsto)$: $\mathrm{FOCat} \times \mathrm{FOCat}$ iff

$$\begin{gathered}\forall\left(\hat{\alpha}, \hat{\beta} : \mathsf{ShuffleComb}\&q\right)(\rho : \mathrm{FOCat}),\\ (\hat{\alpha} \rhd \rho) \land \left(\hat{\alpha} \longrightarrow^{*} \hat{\beta}\right) \Rightarrow \exists(\pi : \mathsf{ShuffleComb}\&q), (\rho \mapsto^{*} \pi) \land \left(\hat{\beta} \rhd \pi\right)\end{gathered} \tag{63}$$

or, with relational composition

$$(\lhd) \;;\; (\longrightarrow^{*}) \subseteq (\mapsto^{*}) \;;\; (\lhd) \tag{64}$$

or, with a diagram (unboxed are assumptions, boxed are results)

$$\begin{array}{ccc} \hat{\alpha} & \rhd & \rho \\ \downarrow_{*} & & \boxed{\downarrow_{*}} \\ \hat{\beta} & \boxed{\rhd} & \boxed{\pi} \end{array} \tag{65}$$

A relation $(\trianglerighteq)$: $\mathrm{ShuffleComb}\&q \times \mathrm{FOCat}$ is a bisimulation for relations $(\longrightarrow)$: $\mathrm{ShuffleComb}\&q \times \mathrm{ShuffleComb}\&q$ and $(\mapsto)$: $\mathrm{FOCat} \times \mathrm{FOCat}$
iff
$(\trianglerighteq)$ is both a simulation and a cosimulation

I strongly believe that $(\trianglerighteq)$ is a bisimulation:

Bisimulation Definition

A relation $(\trianglerighteq)$: $\mathrm{ShuffleComb}\&q \times \mathrm{FOCat}$ is a bisimulation for relations $(\longrightarrow) : \mathrm{ShuffleComb}\&q \times \mathrm{ShuffleComb}\&q$ and $(\mapsto) : \mathrm{FOCat} \times \mathrm{FOCat}$
iff
$(\trianglerighteq)$ is both a simulation and a cosimulation

I strongly believe that $(\trianglerighteq)$ is a bisimulation:
- $(\trianglerighteq)$ is a simulation
  - proved in Rocq—a proof assistant, previously called Coq
- $(\trianglerighteq)$ is a cosimulation
  - strong believes that it is true (no proof yet)

We can extend (▶) and (⊳) to accomodate Higher-Order Concatenative Programs

$$\begin{aligned} C\,I\,q\,\alpha &\longrightarrow^* \alpha\,q \\ C\,B\,q\,\alpha\,x &\longrightarrow^* \alpha\,(q\,x) \\ C\,B\,(C\,I)\,q\,x &\longrightarrow^* q\,(C\,I\,x) \\ C\,(B\,B\,B)\,(C\,B)\,q\,\alpha\,\beta &\longrightarrow^* q\,(C\,B\,\alpha\,\beta) \end{aligned} \qquad \begin{aligned} \texttt{[}\rho\texttt{]}\,\texttt{call} &\mapsto \rho \\ x\,\texttt{[}\rho\texttt{]}\,\texttt{dip} &\mapsto \rho\,x \\ x\,\texttt{unit} &\mapsto \texttt{[}x\texttt{]} \\ \texttt{[}\pi\texttt{]}\,\texttt{[}\rho\texttt{]}\,\texttt{cat} &\mapsto \texttt{[}\,\pi\,\rho\,\texttt{]} \end{aligned} \tag{66}$$

The encoding for quotations that push values and compose quotations:

$$(C\,I\,x)\,q \longrightarrow I\,q\,x \longrightarrow q\,x \tag{67}$$

$$(C\,B\,\alpha\,\beta)\,q \longrightarrow B\,\beta\,\alpha\,q \longrightarrow \beta\,(\alpha\,q) \tag{68}$$

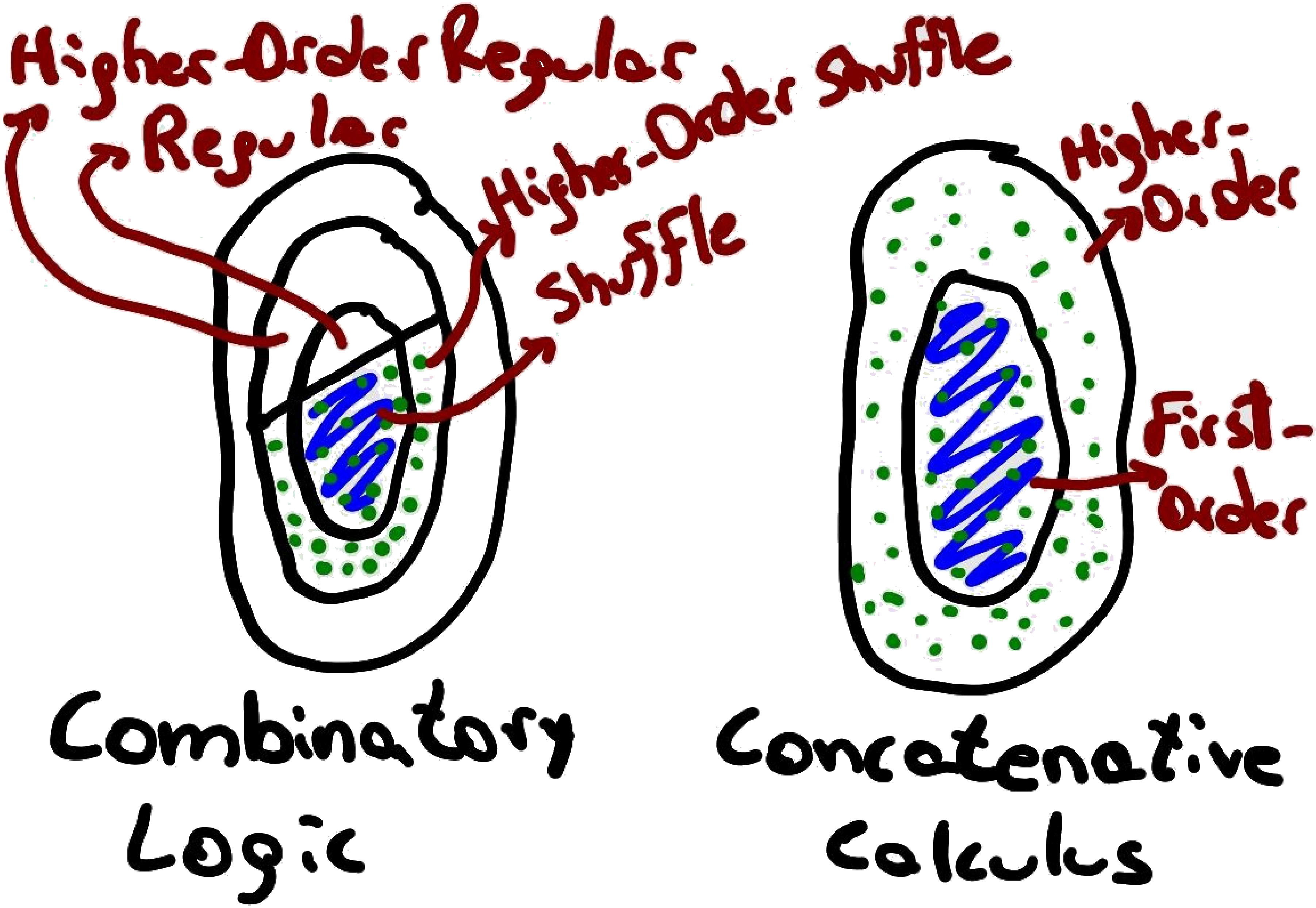
Regular and Shuffle Combinators must keep q at the start, without copies

Higher-order Regular and Shuffle Combinators give some flexibility to that

We consider arguments of a Higher-order Combinator, also a Higher-order Combinator

Thus, q must be "eventually" at the start

$$\begin{gathered} \alpha\,(q\,x)\,y\,z \longrightarrow^* q\,x\,z \\ \alpha_0\,(\alpha_1\,(\dots(\alpha_n\,q\,x_0)\dots)\,x_1)\,x_2\,x_3\dots \longrightarrow^* \alpha_1\,(\dots(\alpha_n\,q\,x_0)\dots)\,x_1\,x_3 \end{gathered} \tag{69}$$


Higher-Order Regular
Regular
Higher-Order Shuffle
Shuffle
Combinatory Logic
Higher-Order
First-Order
Concatenative Calculus