Symmetries in Reversible Programming

From Symmetric Rig Groupoids to Reversible Programming Languages

Logic & Semantics Seminar
May 27, 2022

Computation

- What is Computation?
  - It's the process of calculating functions on natural numbers.
  - There are well-known models of computation:
    - Turing machines
    - Lambda calculi
    - Boolean circuits
- In reality:
  - Computation is whatever a computer does.
  - Computers are implemented as digital/electronic circuits.
  - The basic building blocks are boolean logic gates.
  - They manipulate bits (0s and 1s).
- Shouldn't they follow laws of physics?

The Physical Nature of Computation

- Computation is a physical process (Landauer 1987).
- Computers are physical devices that manipulate bits, they consume energy to do computation, and laws of thermodynamics apply.
- We can use information-theoretic notions of entropy to understand the physical nature of computation.
- Landauer's princple: any logically irreversible manipulation of information increases the entropy of a system.
- A logic gate is simply a boolean function: $\{0,1\}^k \to \{0,1\}$.
- This throws away information about its input, and generates entropy ($kT \ln(2)$ for each bit).
- Conventional models of computation, such as Boolean circuits, Turing machines, λ-calculus, use irreversible primitives for computation.
- We can have logically-reversible models of computation:
  - Reversible Turing machines (Bennett 1970)
  - Reversible Logic gates (Toffoli 1980)
- What is the λ-calculus of reversible computing?

Reversible Computing

- Reversible computing is about programming with sequences of reversible operations, which run on (non-existent) thermodynamically-reversible computers.
- Obviously, we need to:
  - understand their categorical semantics
  - design high-level programming languages for them

Computational View

- Gödel-Church-Turing – Recursive functions, Turing-computable functions, (Untyped) Lambda Calculus programs
- Scwichtenberg, Leivant – Simply-typed lambda calculus, extended polynomials
- Landauer's Principle – computation is a physical process, any logically irreversible manipulation of information increases the entropy of a system.
- Landauer's Limit – there is a theoretical limit to the energy consumption of computation.
- Bennett – Computation in Turing Machines is logically irreversible. Propose Reversible Turing Machines, partially injective functions
- Fredkin, Toffoli – Conservative Logic
- Sabry et. al. – Simply-typed calculus of isomorphisms (Π)
- Today – Π, bijective functions [1]

[1] the name is confusing, also Π^0, or Theseus

Logical View

- Curry, Howard, Scott, Lambek:
  - Simply-typed lambda calculus $\leftrightsquigarrow$ Intuitionistic Logic $\leftrightsquigarrow$ Cartesian Closed Categories
- Linear Logic, Linear lambda calculi $\leftrightsquigarrow$ Seely Categories, Linear-Non-Linear Adjunctions
- Extensional Type Theory $\leftrightsquigarrow$ 1-toposes
- Homotopy Type Theory $\leftrightsquigarrow$ ∞-toposes
- Quantum Computing $\leftrightsquigarrow$ Dagger Symmetric Monoidal Categories
- Today – Π, Symmetric Rig Groupoids

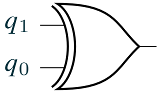
What makes a programming language good?

- The language has models
- The language is logically consistent
- The language is complete with respect to a class of models
- The syntactic and mathematical models (strongly) agree
- Programs have normal forms
- Equivalence of programs is decidable/axiomatisable

I'm going to describe a good reversible programming language.

Boolean Circuits

A boolean gate computes a function $f : \{ 0, 1 \}^k \to \{ 0, 1 \}$.



q_0	q_1	$q_1 + q_0$
0	0	0
0	1	1
1	0	1
1	1	0

Reversible Boolean Circuits: CNOT

A reversible boolean gate computes a permutation $f : \{ 0, 1 \}^k \rightarrow \{ 0, 1 \}^k$.

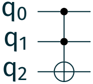
CNOT

q_0	q_1	q_0	$q_1 + q_0$
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 3 & 2 \end{pmatrix}$$

Reversible Boolean Circuits: TOFFOLI/CCNOT

TOFFOLI

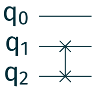


q_0	q_1	q_2	q_0	q_1	$q_2 + q_0 \cdot q_1$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 1 & 2 & 3 & 4 & 5 & 7 & 6 \end{pmatrix}$$

Reversible Boolean Circuits: FREDKIN

FREDKIN



q_0	q_1	q_2	q_0	q_0	$q_2 + q_0 \cdot q_1$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 1 & 2 & 3 & 4 & 5 & 7 & 6 \end{pmatrix}$$

A reversible programming language with finite types: Π

Value types	A, B	$::=$	$\mathbb{0} \mid \mathbb{1} \mid A + B \mid A \times B$
Values	v, w	$::=$	$\mathsf{tt} \mid \mathsf{inj}_1\ v \mid \mathsf{inj}_2\ v \mid (v, w)$
Program types			$A \longleftrightarrow_1 B$
Programs	c	$::=$	

$$
\begin{array}{rrcll}
\mathsf{id}{\leftrightarrow}_1 : & A & \longleftrightarrow_1 & A & : \mathsf{id}{\leftrightarrow}_1 \\
\\
\mathsf{unite_+l} : & \mathbb{0} + A & \longleftrightarrow_1 & A & : \mathsf{uniti_+l} \\
\mathsf{swap_+} : & A + B & \longleftrightarrow_1 & B + A & : \mathsf{swap_+} \\
\mathsf{assocl_+} : & A + (B + C) & \longleftrightarrow_1 & (A + B) + C & : \mathsf{assocr_+} \\
\\
\mathsf{unite{\star}l} : & \mathbb{1} \times A & \longleftrightarrow_1 & A & : \mathsf{uniti{\star}l} \\
\mathsf{swap{\star}} : & A \times B & \longleftrightarrow_1 & B \times A & : \mathsf{swap{\star}} \\
\mathsf{assocl{\star}} : & A \times (B \times C) & \longleftrightarrow_1 & (A \times B) \times C & : \mathsf{assocr{\star}} \\
\\
\mathsf{absorbr} : & \mathbb{0} \times A & \longleftrightarrow_1 & \mathbb{0} & : \mathsf{factorzl} \\
\mathsf{dist} : & (A + B) \times C & \longleftrightarrow_1 & (A \times C) + (B \times C) & : \mathsf{factor}
\end{array}
$$

$$
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : B \longleftrightarrow_1 C}{\vdash c_1 \circledcirc c_2 : A \longleftrightarrow_1 C}
\qquad
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : C \longleftrightarrow_1 D}{\vdash c_1 \oplus c_2 : A + C \longleftrightarrow_1 B + D}
\qquad
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : C \longleftrightarrow_1 D}{\vdash c_1 \otimes c_2 : A \times C \longleftrightarrow_1 B \times D}
$$

Figure: Π syntax

Reversible Boolean Circuits: 3-bit Toffoli gate

$\text{controlled} : (c : A \leftrightarrow_1 A) \to (\mathbb{2} \times A \leftrightarrow_1 \mathbb{2} \times A)$
$\text{controlled}\ c = \text{dist} \circledcirc ((\text{id}{\leftrightarrow_1} \otimes c) \oplus \text{id}{\leftrightarrow_1}) \circledcirc \text{factor}$

$\text{not} : \mathbb{2} \leftrightarrow_1 \mathbb{2}$
$\text{not} = \text{swap}_+$

$\text{cnot} : \mathbb{2} \times \mathbb{2} \leftrightarrow_1 \mathbb{2} \times \mathbb{2}$
$\text{cnot} = \text{controlled not}$

$\text{toffoli}_3 : \mathbb{2} \times (\mathbb{2} \times \mathbb{2}) \leftrightarrow_1 \mathbb{2} \times (\mathbb{2} \times \mathbb{2})$
$\text{toffoli}_3 = \text{controlled cnot}$

Semantics of Π

$$
\begin{array}{lcl}
[\![\mathsf{unite}_{+}\mathsf{l}]\!]\ (\mathsf{inl}\ v) & = & v \\
[\![\mathsf{uniti}_{+}\mathsf{l}]\!]\ v & = & \mathsf{inl}\ v \\
[\![\mathsf{swap}_{+}]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inr}\ v \\
[\![\mathsf{swap}_{+}]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inl}\ v \\
[\![\mathsf{assocl}_{+}]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inl}\ (\mathsf{inl}\ v) \\
[\![\mathsf{assocl}_{+}]\!]\ (\mathsf{inr}\ (\mathsf{inl}\ v)) & = & \mathsf{inl}\ (\mathsf{inr}\ v) \\
[\![\mathsf{assocl}_{+}]\!]\ (\mathsf{inr}\ (\mathsf{inr}\ v)) & = & \mathsf{inr}\ v \\
[\![\mathsf{assocr}_{+}]\!]\ (\mathsf{inl}\ (\mathsf{inl}\ v)) & = & \mathsf{inl}\ v \\
[\![\mathsf{assocr}_{+}]\!]\ (\mathsf{inl}\ (\mathsf{inr}\ v)) & = & \mathsf{inr}\ (\mathsf{inl}\ v) \\
[\![\mathsf{assocr}_{+}]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inr}\ (\mathsf{inr}\ v)
\end{array}
$$

$$
\begin{array}{lcl}
[\![\mathsf{unite}{\star}\mathsf{l}]\!]\ (\star, v) & = & v \\
[\![\mathsf{uniti}{\star}\mathsf{l}]\!]\ v & = & (\star, v) \\
[\![\mathsf{swap}{\star}]\!]\ (v_1, v_2) & = & (v_2, v_1) \\
[\![\mathsf{assocl}{\star}]\!]\ (v_1, (v_2, v_3)) & = & ((v_1, v_2), v_3) \\
[\![\mathsf{assocr}{\star}]\!]\ ((v_1, v_2), v_3) & = & (v_1, (v_2, v_3)) \\
[\![\mathsf{dist}]\!]\ (\mathsf{inl}\ v_1, v_3) & = & \mathsf{inl}\ (v_1, v_3) \\
[\![\mathsf{dist}]\!]\ (\mathsf{inr}\ v_2, v_3) & = & \mathsf{inr}\ (v_2, v_3) \\
[\![\mathsf{factor}]\!]\ (\mathsf{inl}\ (v_1, v_3)) & = & (\mathsf{inl}\ v_1, v_3) \\
[\![\mathsf{factor}]\!]\ (\mathsf{inr}\ (v_2, v_3)) & = & (\mathsf{inr}\ v_2, v_3) \\
[\![\mathsf{id}{\leftrightarrow}_1]\!]\ v & = & v
\end{array}
$$

$$
\begin{array}{lcl}
[\![(c_1 \circledcirc c_2)]\!]\ v & = & [\![c_2]\!] ([\![c_1]\!]\ v) \\
[\![(c_1 \oplus c_2)]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inl}\ ([\![c_1]\!]\ v) \\
[\![(c_1 \oplus c_2)]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inr}\ ([\![c_2]\!]\ v) \\
[\![(c_1 \otimes c_2)]\!]\ (v_1, v_2) & = & ([\![c_1]\!]\ v_1, [\![c_2]\!]\ v_2)
\end{array}
$$

Semantics of Π

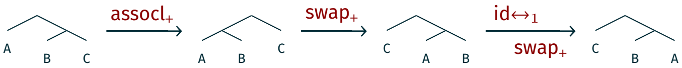
$$
\begin{array}{lll}
[\![\,\mathsf{unite_{+}l}\,]\!]\ (\mathsf{inl}\ v) & = & v \\
[\![\,\mathsf{uniti_{+}l}\,]\!]\ v & = & \mathsf{inl}\ v \\
[\![\,\mathsf{swap_{+}}\,]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inr}\ v \\
[\![\,\mathsf{swap_{+}}\,]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inl}\ v \\
[\![\,\mathsf{assocl_{+}}\,]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inl}\ (\mathsf{inl}\ v) \\
[\![\,\mathsf{assocl_{+}}\,]\!]\ (\mathsf{inr}\ (\mathsf{inl}\ v)) & = & \mathsf{inl}\ (\mathsf{inr}\ v) \\
[\![\,\mathsf{assocl_{+}}\,]\!]\ (\mathsf{inr}\ (\mathsf{inr}\ v)) & = & \mathsf{inr}\ v \\
[\![\,\mathsf{assocr_{+}}\,]\!]\ (\mathsf{inl}\ (\mathsf{inl}\ v)) & = & \mathsf{inl}\ v \\
[\![\,\mathsf{assocr_{+}}\,]\!]\ (\mathsf{inl}\ (\mathsf{inr}\ v)) & = & \mathsf{inr}\ (\mathsf{inl}\ v) \\
[\![\,\mathsf{assocr_{+}}\,]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inr}\ (\mathsf{inr}\ v)
\end{array}
$$

$$
\begin{array}{lll}
[\![\,\mathsf{unite{\star}l}\,]\!]\ (\star, v) & = & v \\
[\![\,\mathsf{uniti{\star}l}\,]\!]\ v & = & (\star, v) \\
[\![\,\mathsf{swap{\star}}\,]\!]\ (v_1, v_2) & = & (v_2, v_1) \\
[\![\,\mathsf{assocl{\star}}\,]\!]\ (v_1, (v_2, v_3)) & = & ((v_1, v_2), v_3) \\
[\![\,\mathsf{assocr{\star}}\,]\!]\ ((v_1, v_2), v_3) & = & (v_1, (v_2, v_3)) \\
[\![\,\mathsf{dist}\,]\!]\ (\mathsf{inl}\ v_1, v_3) & = & \mathsf{inl}\ (v_1, v_3) \\
[\![\,\mathsf{dist}\,]\!]\ (\mathsf{inr}\ v_2, v_3) & = & \mathsf{inr}\ (v_2, v_3) \\
[\![\,\mathsf{factor}\,]\!]\ (\mathsf{inl}\ (v_1, v_3)) & = & (\mathsf{inl}\ v_1, v_3) \\
[\![\,\mathsf{factor}\,]\!]\ (\mathsf{inr}\ (v_2, v_3)) & = & (\mathsf{inr}\ v_2, v_3) \\
[\![\,\mathsf{id}{\leftrightarrow_1}\,]\!]\ v & = & v
\end{array}
$$

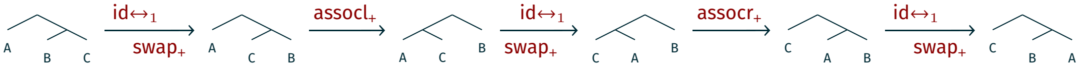
$$
\begin{array}{lll}
[\![(c_1 \circledcirc c_2)]\!]\ v & = & [\![\,c_2\,]\!]([\![\,c_1\,]\!]\ v) \\
[\![(c_1 \oplus c_2)]\!]\ (\mathsf{inl}\ v) & = & \mathsf{inl}\ ([\![\,c_1\,]\!]\ v) \\
[\![(c_1 \oplus c_2)]\!]\ (\mathsf{inr}\ v) & = & \mathsf{inr}\ ([\![\,c_2\,]\!]\ v) \\
[\![(c_1 \otimes c_2)]\!]\ (v_1, v_2) & = & ([\![\,c_1\,]\!]\ v_1, [\![\,c_2\,]\!]\ v_2)
\end{array}
$$

Does it compute every bijection?

Permutations as tree transformations

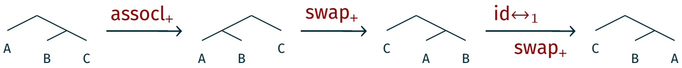


p_1

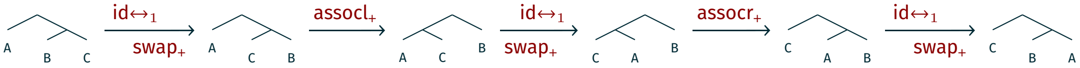


p_2

Permutations as tree transformations



p_1



p_2

Can we find a sound and complete set of equations to decide when two programs are equal?

Equational theory: Examples of 2-combinators

$$\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : A \longleftrightarrow_1 B}{\vdash c_1 \longleftrightarrow_2 c_2}$$

Equational theory: Examples of 2-combinators

$$\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : A \longleftrightarrow_1 B}{\vdash c_1 \longleftrightarrow_2 c_2}$$

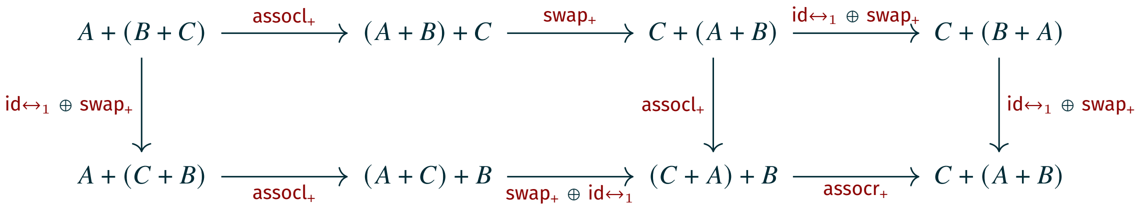
$$\begin{array}{rrcll}
\mathsf{id}{\leftrightarrow_2} : & c & \longleftrightarrow_2 & c & : \mathsf{id}{\leftrightarrow_2} \\
\mathsf{assoc}{\circledcirc}\mathsf{l} : & c_1 \circledcirc (c_2 \circledcirc c_3) & \longleftrightarrow_2 & (c_1 \circledcirc c_2) \circledcirc c_3 & : \mathsf{assoc}{\circledcirc}\mathsf{r} \\
\mathsf{idl}{\circledcirc}\mathsf{l} : & \mathsf{id}{\leftrightarrow_1} \circledcirc c & \longleftrightarrow_2 & c & : \mathsf{idl}{\circledcirc}\mathsf{r} \\
\mathsf{idr}{\circledcirc}\mathsf{l} : & c \circledcirc \mathsf{id}{\leftrightarrow_1} & \longleftrightarrow_2 & c & : \mathsf{idr}{\circledcirc}\mathsf{r} \\
\mathsf{linv}{\circledcirc}\mathsf{l} : & c \circledcirc !{\leftrightarrow_1} c & \longleftrightarrow_2 & \mathsf{id}{\leftrightarrow_1} & : \mathsf{linv}{\circledcirc}\mathsf{r} \\
\mathsf{swapl}_+{\leftrightarrow_2} : & \mathsf{swap}_+ \circledcirc (c_1 \oplus c_2) & \longleftrightarrow_2 & (c_2 \oplus c_1) \circledcirc \mathsf{swap}_+ & : \mathsf{swapr}_+{\leftrightarrow_2}
\end{array}$$

$$\begin{array}{rrcll}
\mathsf{pentagon}_+\mathsf{l} : & \mathsf{assocr}_+ \circledcirc \mathsf{assocr}_+ & \longleftrightarrow_2 & ((\mathsf{assocr}_+ \oplus \mathsf{id}{\leftrightarrow_1}) \circledcirc \mathsf{assocr}_+) \circledcirc (\mathsf{id}{\leftrightarrow_1} \oplus \mathsf{assocr}_+) & : \mathsf{pentagon}_+\mathsf{r} \\
\mathsf{hexagonl}_+\mathsf{l} : & (\mathsf{assocl}_+ \circledcirc \mathsf{swap}_+) \circledcirc \mathsf{assocl}_+ & \longleftrightarrow_2 & ((\mathsf{id}{\leftrightarrow_1} \oplus \mathsf{swap}_+) \circledcirc \mathsf{assocl}_+) \circledcirc (\mathsf{swap}_+ \oplus \mathsf{id}{\leftrightarrow_1}) & : \mathsf{hexagonl}_+\mathsf{r}
\end{array}$$

$$\frac{\vdash \alpha_1 : c_1 \longleftrightarrow_2 c_2 \quad \vdash \alpha_2 : c_2 \longleftrightarrow_2 c_3}{\vdash \alpha_1 \blacksquare \alpha_2 : c_1 \longleftrightarrow_2 c_3} \qquad \frac{\vdash \alpha_1 : c_1 \longleftrightarrow_2 c_3 \quad \vdash \alpha_2 : c_2 \longleftrightarrow_2 c_4}{\vdash \alpha_1 \boxdot \alpha_2 : (c_1 \circledcirc c_2) \longleftrightarrow_2 (c_3 \circledcirc c_4)}$$

Solving the example

```
p₁₂ : p₁ ↔₂ p₂
p₁₂ = assoc◎l
   ■ ((idr◎r ■ (id↔₂ ⊡ linv◎r) ■ assoc◎l ■ (hexagonl₊l ⊡ id↔₂))
     ⊡ (idl◎r ■ ( linv◎r  ⊡ id↔₂)))
   ■ ((id↔₂ ⊡ (linv◎l ⊡ id↔₂)) ■ (id↔₂ ⊡ idl◎l))
   ■ assoc◎r ■ assoc◎r ■ assoc◎r
```



This language is too complicated, let’s start from a very simple language.

A reversible language with 2 bits, Π_2

Value types	A, B	$::=$	$\mathbb{2}$
Values	v, w	$::=$	$\mathsf{ff} \mid \mathsf{tt}$
Program types			$A \longleftrightarrow_1 B$
Programs	c	$::=$	

$$\mathsf{id}{\leftrightarrow}_1 : \quad A \quad \longleftrightarrow_1 \quad A \quad : \mathsf{id}{\leftrightarrow}_1$$

$$\mathsf{swap}_2 : \quad \mathbb{2} \quad \longleftrightarrow_1 \quad \mathbb{2} \quad : \mathsf{swap}_+$$

$$\frac{\vdash c_1 : A \longleftrightarrow_1 B \qquad \vdash c_2 : B \longleftrightarrow_1 C}{\vdash c_1 \circledcirc c_2 : A \longleftrightarrow_1 C}$$

$$\begin{array}{rcccl}
\mathsf{id}{\leftrightarrow}_2 : & c & \longleftrightarrow_2 & c & : \mathsf{id}{\leftrightarrow}_2 \\
\mathsf{assoc}{\circledcirc}\mathsf{l} : & c_1 \circledcirc (c_2 \circledcirc c_3) & \longleftrightarrow_2 & (c_1 \circledcirc c_2) \circledcirc c_3 & : \mathsf{assoc}{\circledcirc}\mathsf{r} \\
\mathsf{idl}{\circledcirc}\mathsf{l} : & \mathsf{id}{\leftrightarrow}_1 \circledcirc c & \longleftrightarrow_2 & c & : \mathsf{idl}{\circledcirc}\mathsf{r} \\
\mathsf{idr}{\circledcirc}\mathsf{l} : & c \circledcirc \mathsf{id}{\leftrightarrow}_1 & \longleftrightarrow_2 & c & : \mathsf{idr}{\circledcirc}\mathsf{r} \\
\mathsf{linv}{\circledcirc}\mathsf{l} : & c \circledcirc !{\leftrightarrow}_1 c & \longleftrightarrow_2 & \mathsf{id}{\leftrightarrow}_1 & : \mathsf{linv}{\circledcirc}\mathsf{r} \\
\mathsf{swap}_2{\circledcirc}\mathsf{swap}_2 : & \mathsf{swap}_2 \circledcirc \mathsf{swap}_2 & \longleftrightarrow_2 & \mathsf{id}{\leftrightarrow}_1 & : \mathsf{swap}_2{\circledcirc}\mathsf{swap}_2
\end{array}$$

$$\frac{\vdash \alpha_1 : c_1 \longleftrightarrow_2 c_2 \qquad \vdash \alpha_2 : c_2 \longleftrightarrow_2 c_3}{\vdash \alpha_1 \blacksquare \alpha_2 : c_1 \longleftrightarrow_2 c_3} \qquad \frac{\vdash \alpha_1 : c_1 \longleftrightarrow_2 c_3 \qquad \vdash \alpha_2 : c_2 \longleftrightarrow_2 c_4}{\vdash \alpha_1 \boxdot \alpha_2 : (c_1 \circledcirc c_2) \longleftrightarrow_2 (c_3 \circledcirc c_4)}$$

Semantics of Π_2

- Obviously, there should only be two reversible programs on $\mathbb{2}$, upto equivalence.
- For every $p : \mathbb{2} \longleftrightarrow_1 \mathbb{2}$, either $p \longleftrightarrow_2 \mathsf{id}{\leftrightarrow}_1$ or $p \longleftrightarrow_2 \mathsf{swap}_2$.
- How can we prove it?

Semantics of Π_2

- Idea: we're simply describing the groupoid $\mathcal{B}\mathbb{Z}_2$, where $\mathbb{Z}_2$ is the automorphism group of $\mathbb{2}$.
- In HoTT, automorphism groups and their deloopings have a special encoding.
- The automorphism group of T is $\mathsf{Aut}(T) \triangleq T \simeq T$.
- The delooping of the automorphism group of T is $\mathcal{B}\mathsf{Aut}(T) \triangleq \sum_{X:\mathcal{U}} \|X =_{\mathcal{U}} T\|_{-1}$.

Proposition

- $\pi_1 : \mathcal{B}\mathsf{Aut}(T) \to \mathcal{U}$ is a univalent fibration.
- If T is an n-type, then $\mathcal{B}\mathsf{Aut}(T)$ is an (n+1)-type.
- $\mathcal{B}\mathsf{Aut}(T)$ is 0-connected.
- $\Omega(\mathcal{B}\mathsf{Aut}(T), T_0 \equiv (T, \mathsf{refl})) \simeq \mathsf{Aut}(T)$.

Semantics of Π_2

- Now, we can use this to do NbE.
- Interpret 1-combinators as 1-paths in $\mathcal{B}\mathsf{Aut}(\mathbb{2})$.
- Interpret 2-combinators as 2-paths in $\mathcal{B}\mathsf{Aut}(\mathbb{2})$.
- For every $p : (\mathbb{2}_0 =_{\mathcal{B}\mathsf{Aut}(\mathbb{2})} \mathbb{2}_0)$, either $p = \mathsf{refl}$, or $p = \mathsf{ua}(\mathsf{not} : \mathbb{2} \simeq \mathbb{2})$.

What really happened?

$$\Omega(\mathcal{B}\mathsf{Aut}(\mathbb{2}), \mathbb{2}_0) \simeq \mathsf{Aut}(\mathbb{2}) \simeq \mathbb{2}$$

But,

$$\begin{aligned} \mathsf{Aut}(\mathbb{3}) &\simeq \mathbb{6} \\ \mathsf{Aut}(\mathbb{4}) &\simeq \mathbb{24} \\ &\ldots \\ \mathsf{Aut}([n]) &\simeq [n!] \end{aligned}$$

How do we go to n-bit languages?

Several steps

We will go from Π to $\mathcal{U}_{\mathsf{Fin}}$, the groupoid of finite sets and bijections, and back.

$$\Pi \simeq \Pi^{+} \simeq \Pi^{\wedge} \simeq \sqcup_n \mathcal{B}\mathsf{S_n} \simeq \sqcup_n \mathcal{B}\mathcal{L}_n \simeq \sqcup_n \mathcal{B}\mathsf{Aut}(\mathsf{Fin_n}) \simeq \mathcal{U}_{\mathsf{Fin}}$$

Π, or $\mathcal{F}_{\mathrm{SR}}(0)$

Value types	A, B	$::=$	$\mathbb{0} \mid \mathbb{1} \mid A + B \mid A \times B$
Values	v, w	$::=$	$\mathsf{tt} \mid \mathsf{inj}_1\ v \mid \mathsf{inj}_2\ v \mid (v, w)$
Program types			$A \longleftrightarrow_1 B$
Programs	c	$::=$	

$$
\begin{array}{rrcll}
\mathsf{id}{\leftrightarrow_1} : & A & \longleftrightarrow_1 & A & : \mathsf{id}{\leftrightarrow_1} \\
\\
\mathsf{unite_+l} : & \mathbb{0} + A & \longleftrightarrow_1 & A & : \mathsf{uniti_+l} \\
\mathsf{swap_+} : & A + B & \longleftrightarrow_1 & B + A & : \mathsf{swap_+} \\
\mathsf{assocl_+} : & A + (B + C) & \longleftrightarrow_1 & (A + B) + C & : \mathsf{assocr_+} \\
\\
\mathsf{unite{\star}l} : & \mathbb{1} \times A & \longleftrightarrow_1 & A & : \mathsf{uniti{\star}l} \\
\mathsf{swap{\star}} : & A \times B & \longleftrightarrow_1 & B \times A & : \mathsf{swap{\star}} \\
\mathsf{assocl{\star}} : & A \times (B \times C) & \longleftrightarrow_1 & (A \times B) \times C & : \mathsf{assocr{\star}} \\
\\
\mathsf{absorbr} : & \mathbb{0} \times A & \longleftrightarrow_1 & \mathbb{0} & : \mathsf{factorzl} \\
\mathsf{dist} : & (A + B) \times C & \longleftrightarrow_1 & (A \times C) + (B \times C) & : \mathsf{factor}
\end{array}
$$

$$
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : B \longleftrightarrow_1 C}{\vdash c_1 \circledcirc c_2 : A \longleftrightarrow_1 C}
\qquad
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : C \longleftrightarrow_1 D}{\vdash c_1 \oplus c_2 : A + C \longleftrightarrow_1 B + D}
\qquad
\frac{\vdash c_1 : A \longleftrightarrow_1 B \quad \vdash c_2 : C \longleftrightarrow_1 D}{\vdash c_1 \otimes c_2 : A \times C \longleftrightarrow_1 B \times D}
$$

The 2-combinators are: groupoid laws, naturality, and coherence conditions of symmetric rig categories.

Π^+ or $\mathcal{F}_{\text{SM}}(1)$

$$
\begin{array}{llll}
\textit{Value types} & A, B & ::= & \mathbb{0} \mid \mathbb{1} \mid A + B \\
\textit{Values} & v, w & ::= & \mathsf{tt} \mid \mathsf{inj}_1\ v \mid \mathsf{inj}_2\ v \\
\textit{Program types} & & & A \longleftrightarrow_1 B \\
\textit{Programs} & c & ::= &
\end{array}
$$

$$
\begin{array}{rrcll}
\mathsf{id}{\leftrightarrow}_1 : & A & \leftrightarrow^+ & A & : \mathsf{id}{\leftrightarrow}_1 \\
\mathsf{unite_+l} : & \mathbb{0} + A & \leftrightarrow^+ & A & : \mathsf{uniti_+l} \\
\mathsf{swap_+} : & A + B & \leftrightarrow^+ & B + A & : \mathsf{swap_+} \\
\mathsf{assocl_+} : & A + (B + C) & \leftrightarrow^+ & (A + B) + C & : \mathsf{assocr_+}
\end{array}
$$

$$
\frac{\vdash c_1 : A \leftrightarrow^+ B \quad \vdash c_2 : B \leftrightarrow^+ C}{\vdash c_1 \circledcirc c_2 : A \leftrightarrow^+ C}
\qquad
\frac{\vdash c_1 : A \leftrightarrow^+ B \quad \vdash c_2 : C \leftrightarrow^+ D}{\vdash c_1 \oplus c_2 : A + C \leftrightarrow^+ B + D}
$$

The 2-combinators are: groupoid laws, naturality, and coherence conditions of symmetric monoidal categories.

Step 1: Π to Π^+

- We show that Π^+ is a symmetric rig, then construct a symmetric rig functor $\Pi \longrightarrow \Pi^+$.
- On types, build multiplication using iterated addition:

$$\begin{aligned} \mathbb{0} \times Y &\triangleq \mathbb{0} \\ \mathbb{1} \times Y &\triangleq Y \\ (X_1 + X_2) \times Y &\triangleq X_1 \times Y + X_2 \times Y \end{aligned}$$

- On 1-combinators, use distributivity.
- To go back, use Laplaza coherence conditions for distributivity.

- We get a symmetric rig equivalence: $\Pi \simeq \Pi^+$.

$\Pi^{\wedge}$, or the minimal PROP

$$\begin{array}{llll} \textit{Value types} & A, B & ::= & \mathbb{0} \mid \mathsf{S}\,A \\ \textit{Values} & v, w & ::= & \mathsf{tt} \mid \mathsf{S}\,v \\ \textit{Program types} & & & A \longleftrightarrow_1 B \\ \textit{Programs} & c & ::= & \end{array}$$

$$\begin{array}{rcccl} \mathsf{id}{\leftrightarrow_1} : & n & \leftrightarrow^{\wedge} & n & : \mathsf{id}{\leftrightarrow_1} \\ \mathsf{swap} : & \mathsf{S}\,\mathsf{S}\,n & \leftrightarrow^{\wedge} & \mathsf{S}\,\mathsf{S}\,n & : \mathsf{swap} \end{array}$$

$$\frac{\vdash c_1 : n \leftrightarrow^{\wedge} m \qquad \vdash c_2 : m \leftrightarrow^{\wedge} o}{\vdash c_1 \circledcirc c_2 : n \leftrightarrow^{\wedge} o} \qquad \frac{\vdash c : n \leftrightarrow^{\wedge} m}{\vdash \oplus(c) : \mathsf{S}\,n \leftrightarrow^{\wedge} \mathsf{S}\,m}$$

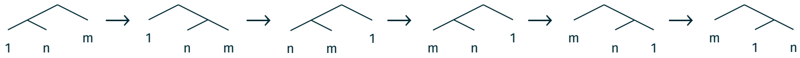
The 2-combinators are: groupoid laws, naturality, a symmetry, and a minimal hexagon.

$$\begin{array}{rccl} \mathsf{swap}^2 : & \mathsf{swap} \circledcirc \mathsf{swap} \longleftrightarrow_2 \mathsf{id}{\leftrightarrow_1} & & : \mathsf{swap}^2 \\ \mathsf{hexagon} : & \mathsf{swap} \circledcirc \mathsf{S}\,\mathsf{swap} \circledcirc \mathsf{S}\,\mathsf{swap} \longleftrightarrow_2 \mathsf{S}\,\mathsf{swap} \circledcirc \mathsf{swap} \circledcirc \mathsf{S}\,\mathsf{swap} & & : \mathsf{hexagon} \end{array}$$

$\Pi^\wedge$, or the minimal PROP

This is symmetric monoidal, small swaps inductively generate big swaps.

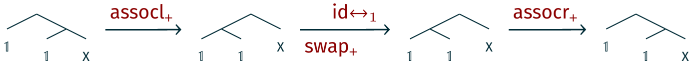
- $\alpha : (n + m) + o \longleftrightarrow_1 n + (m + o)$
- $\lambda : 0 + n \longleftrightarrow_1 n$
- $\rho : n + 0 \longleftrightarrow_1 n$
- $\beta : n + m \longleftrightarrow_1 m + n$
- triangle, pentagon, and (big) hexagon



Hence we get $\Pi^+ \to \Pi^\wedge$.

Step 2: Π^+ to $\Pi^\wedge$

To go back, use a big swap to produce a small swap (an adjacent transposition).



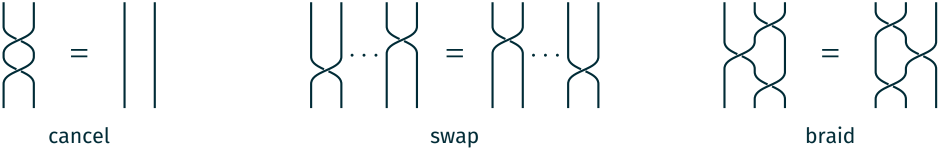
This gives a symmetric monoidal equivalence $\Pi \simeq \Pi^+$.

Step 3: $\Omega(\Pi^\wedge, n)$

- Now, the loopspace of $\Pi^\wedge$ at each $n : \mathbb{N}$, $\Omega(\Pi^\wedge, n)$ has all the permutations, upto coherences.
- We will show that the loopspace of $\Pi^\wedge$ at $n : \mathbb{N}$ is equivalent to a presentation of the symmetric group S_n.
- The generators will be sequences of adjacent transpositions, and relations will be Coxeter relations.

Step 3: $\Omega(\Pi^\wedge, n)$ to S_n

The generators are encoded as $\text{List}(\text{Fin}_n)$, and the relations are on $\text{List}(\text{Fin}_n)$.



- Swapping the same two elements twice in a row does nothing.
- When swapping two distinct pairs of elements, the order of swapping doesn't matter.
- Two ways of swapping the first and last elements in a sequence of three elements are the same.

They're not directed, but we can tweak them.

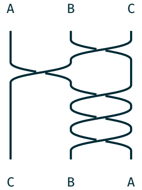
Step 3: $\Omega(\Pi^\wedge, n)$ to S_n

$\rightsquigarrow$: $\mathsf{List}(\mathsf{Fin}_n) \to \mathsf{List}(\mathsf{Fin}_n) \to \mathcal{U}$

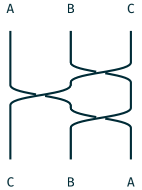
$$\begin{aligned}
\mathsf{cancel}^{\blacklozenge} &: \forall n, l, r \to (l +\!\!+ n :: n :: r) \rightsquigarrow (l +\!\!+ r) \\
\mathsf{swap}^{\blacklozenge} &: \forall n, k, l, r \to (\mathsf{S}\,k < n) \to (l +\!\!+ n :: k :: r) \rightsquigarrow (l +\!\!+ k :: n +\!\!+ r) \\
\mathsf{braid}^{\blacklozenge} &: \forall n, k, l, r \to (l +\!\!+ (n \swarrow 2 + k) +\!\!+ (1 + k + n) :: r) \rightsquigarrow (l +\!\!+ (k + n) :: (n \swarrow 2 + k) +\!\!+ r)
\end{aligned}$$

- This gives a confluent and terminating rewriting system.
- For every w, there exists a unique normal form v such that $w \overset{*}{\rightsquigarrow} v$.
- We get a unique choice function $\mathfrak{nf} : \mathsf{List}(\mathsf{Fin}_n) \to \mathsf{List}(\mathsf{Fin}_n)$.
- For all $l : \mathsf{List}(\mathsf{Fin}_n)$, we have that $l \overset{*}{\rightsquigarrow} \mathfrak{nf}[l]$.
- We can choose a word in each equivalence class, giving $\mathsf{S}_n \triangleq \mathsf{List}(\mathsf{Fin}_n)/\overset{*}{\leftrightsquigarrow} \simeq \mathrm{im}(\mathfrak{nf})$.

Example: permutations as braid diagrams

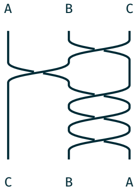


$p_1 \mapsto [1, 0, 1, 1, 1]$

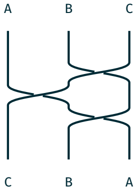


$p_2 \mapsto [1, 0, 1]$

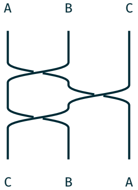
Example: permutations as braid diagrams



$p_1 \mapsto [1, 0, 1, 1, 1]$



$p_2 \mapsto [1, 0, 1]$



Normal form: $[0, 1, 0]$

Step 4: S_n to Lehmer(n)

These normal forms are precisely permutation codes in a factorial number system.

Lehmer : $\mathbb{N} \to \mathcal{U}$

$$\mathsf{Lehmer}(0) \triangleq \mathsf{Fin}_{\mathsf{s}\,0}$$
$$\mathsf{Lehmer}(\mathsf{S}\,n) \triangleq \mathsf{Lehmer}(n) \times \mathsf{Fin}_{\mathsf{s}\,\mathsf{s}\,n}$$

$\mathfrak{em}_n$: Lehmer$(n) \to$ List$(\mathsf{Fin}_{\mathsf{s}\,n})$

$$\mathfrak{em}_0(0) \triangleq \mathsf{nil}$$
$$\mathfrak{em}_{\mathsf{S}\,n}((r, l)) \triangleq \mathfrak{em}_n(l) \mathbin{+\!\!+} ((\mathsf{S}\,n - r) \swarrow r)$$

- $\mathfrak{em}_n$: Lehmer$(n) \to \mathsf{im}(\mathfrak{nf})$ has contractible fibers.
- $\mathsf{S}_n \simeq \mathsf{im}(\mathfrak{nf}) \simeq \mathsf{Lehmer}(n)$.

Transpositions to Permutations

$$(a\ b\ c) \quad \mapsto \quad (b\ a\ c) \quad \mapsto \quad (b\ c\ a) \quad \mapsto \quad (c\ b\ a)$$

$$0 \qquad\qquad 1 \qquad\qquad 0$$

Transpositions to Permutations

$$\begin{array}{ccccccc} (a\ b\ c) & \mapsto & (b\ a\ c) & \mapsto & (b\ c\ a) & \mapsto & (c\ b\ a) \\ & 0 & & 1 & & 0 & \end{array}$$

This produces a Lehmer code: $(0\ 1\ 2)$

Transpositions to Permutations

$$(a\ b\ c) \quad \mapsto \quad (b\ a\ c) \quad \mapsto \quad (b\ c\ a) \quad \mapsto \quad (c\ b\ a)$$

$$0 \qquad\qquad 1 \qquad\qquad 0$$

This produces a Lehmer code: $(0\ 1\ 2)$

$$\begin{array}{ll} 0 & (a\ b\ c) \mapsto (a\ b\ c) \\ 1 & (a\ b\ c) \mapsto (b\ a\ c) \\ 2 & (b\ a\ c) \mapsto (c\ b\ a) \end{array}$$

Step 5: Lehmer(n) to Aut($\mathsf{Fin}_{\mathsf{s}\,\mathsf{n}}$)

Finally, run the Lehmer code to get:

$$\mathsf{S}_\mathsf{n} \simeq \mathsf{im}(\mathfrak{nf}) \simeq \mathsf{Lehmer}(n) \simeq \mathsf{Aut}(\mathsf{Fin}_{\mathsf{s}\,\mathsf{n}})$$

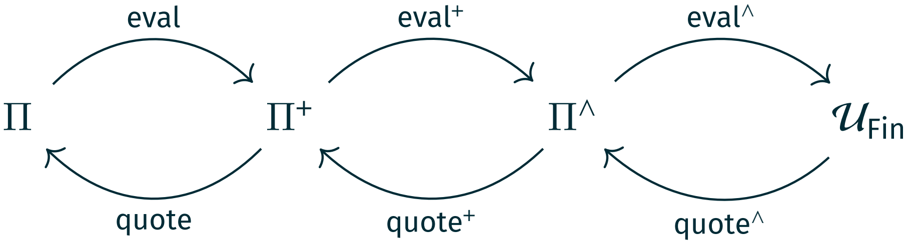
$\mathcal{U}_{\mathsf{Fin}}$

- $\mathcal{U}_{\mathsf{Fin}} \equiv \sum_{X:\mathcal{U}} \sum_{n:\mathbb{N}} \left\|X = \mathsf{Fin}_n\right\|$ is a 1-groupoid.
- $\mathcal{U}_{\mathsf{Fin}_n} \equiv \sum_{X:\mathcal{U}} \left\|X = \mathsf{Fin}_n\right\|$ is a pointed, 0-connected, 1-groupoid, for every $n : \mathbb{N}$.
- $\mathcal{U}_{\mathsf{Fin}} \simeq \sum_{X:\mathcal{U}} \sum_{n:\mathbb{N}} \left\|X = \mathsf{Fin}_n\right\| \simeq \sum_{n:\mathbb{N}} \sum_{X:\mathcal{U}} \left\|X = \mathsf{Fin}_n\right\|$
- $\pi_1 : \mathcal{U}_{\mathsf{Fin}} \to \mathcal{U}$ is a univalent fibration.
- $\Omega(\mathcal{U}_{\mathsf{Fin}_n}, \mathsf{Fin}_n) \simeq \mathsf{Aut}(\mathsf{Fin}_n)$.

Finally, we get the equivalence:

$$\Pi \simeq \Pi^{+} \simeq \Pi^{\wedge} \simeq \mathcal{U}_{\mathsf{Fin}}$$

Normalisation by Evaluation



$$2 \times (2 \times 2) \xrightarrow{\text{eval}} (2 + 2) + (2 + 2) \xrightarrow{\text{eval}^{+}} 8 \xrightarrow{\text{eval}^{\wedge}} \mathsf{Fin}_8$$

Summary

- Curry-Howard-Lambek correspondence:

  Reversible Programming Languages $\leftrightsquigarrow$ Symmetric Rig Groupoids

Π	$\bigsqcup_n \mathcal{B}S_n$	$\mathcal{B}$	$\mathcal{U}_{Fin}$
Types	Natural numbers	Finite sets	0-cells
1-combinators	Generators of S_n	Bijections	1-paths
2-combinators	Relations of S_n	Homotopies	2-paths

- Full-abstraction and adequacy with respect to operational semantics
- Normalisation, equivalence, and synthesis for reversible circuits
- Transfer of theorems about permutations between different representations
- Agda formalisation using HoTT-Agda
  - vikraman/2DTypes, vikraman/popl22-symmetries-artifact

https://dl.acm.org/doi/10.1145/3498667

Future Work

- Construction of the free symmetric monoidal groupoid over a groupoid.
  - See: https://arxiv.org/abs/2110.05412
- Generalised Species of Structures over Groupoids and its differential structure.
- Groupoid models of Linear logic
- Construction of A_n/E_n operads in HoTT.