# Representing, Manipulating and Optimizing Reversible Circuits

Jacques Carette

McMaster University carette@mcmaster.ca

Amr Sabry
Indiana University
sabry@indiana.edu

#### **Abstract**

We show how a typed set of combinators for reversible computations, corresponding exactly to the semiring of permutations, is a convenient basis for representing and manipulating reversible circuits. A categorical interpretation also leads to optimization combinators, and we demonstrate their utility through an example.

#### 1. Introduction

Quantum Computing. Quantum physics differs from classical physics in many ways:

- Superpositions
- Entanglement
- Unitary evolution
- · Composition uses tensor products
- Non-unitary measurement

Quantum Computing & Programming Languages.

- It is possible to adapt all at once classical programming languages to quantum programming languages.
- · Some excellent examples discussed in this workshop
- This assumes that classical programming languages (and implicitly classical physics) can be smoothly adapted to the quantum world.
- There are however what appear to be fundamental differences between the classical and quantum world that make them incompatible
- Let us re-think classical programming foundations before jumping to the quantum world.

Resource-Aware Classical Computing.

 The biggest questionable assumption of classical programming is that it is possible to freely copy and discard information

- A classical programming language which respects no-cloning and no-discarding is the right foundation for an eventual quantum extension
- We want these properties to be inherent in the language; not an afterthought filtered by a type system
- We want to program with isomorphisms or equivalences
- The simplest instance is permutations between finite types which happens to correspond to reversible circuits.

Representing Reversible Circuits: truth table, matrix, reed muller expansion, product of cycles, decision diagram, etc.

any easy way to reproduce Figure 4 on p.7 of Saeedi and Markov? important remark: these are all *Boolean* circuits! Most important part: reversible circuits are equivalent to permutations.

A (Foundational) Syntactic Theory. Ideally, want a notation that

- 1. is easy to write by programmers
- 2. is easy to mechanically manipulate
- 3. can be reasoned about
- 4. can be optimized.

Start with a *foundational* syntactic theory on our way there:

- 1. easy to explain
- 2. clear operational rules
- 3. fully justified by the semantics
- 4. sound and complete reasoning
- 5. sound and complete methods of optimization

A Syntactic Theory. Ideally want a notation that is easy to write by programmers and that is easy to mechnically manipulate for reasoning and optimizing of circuits.

Syntactic calculi good. Popular semantics: Despite the increasing importance of formal methods to the computing industry, there has been little advance to the notion of a "popular semantics" that can be explained to *and used* effectively (for example to optimize or simplify programs) by non-specialists including programmers and first-year students. Although the issue is by no means settled, syntactic theories are one of the candidates for such a popular semantics for they require no additional background beyond knowledge of the programming language itself, and they provide a direct support for the equational reasoning underlying many program transformations.

## 2. Typed Isomorphisms

First, a universe of (finite) types

data U : Set where ZERO : U

```
ONE : U

PLUS : U \rightarrow U \rightarrow U

TIMES : U \rightarrow U \rightarrow U

and its interpretation

[_]: U \rightarrow Set

[ZERO] = \bot

[ONE] = \top

[PLUS t_1 t_2] = [t_1] \uplus [t_2]

[TIMES t_1 t_2] = [t_1] \times [t_2]
```

Equivalences and semirings. If we denote type equivalence by  $\simeq$ , then we can prove that

**Theorem 1.** The collection of all types (Set) forms a commutative semiring (up to  $\simeq$ ).

We also get

**Theorem 2.** If  $A \simeq \text{Fin} m$ ,  $B \simeq \text{Fin} n$  and  $A \simeq B$  then  $m \equiv n$ .

(whose constructive proof is quite subtle).

**Theorem 3.** If  $A \simeq \text{Fin} m$  and  $B \simeq \text{Fin} n$ , then the type of all equivalences  $A \simeq B$  is equivalent to the type of all permutations Perm n.

Equivalences and semirings II. Semiring structures abound. We can define them on:

- 1. equivalences (disjoint union and cartesian product)
- 2. permutations (disjoint union and tensor product)

The point, of course, is that they are related:

**Theorem 4.** The equivalence of Theorem 3 is an isomorphism between the semirings of equivalences of finite types, and of permutations.

A more evocative phrasing might be:

#### Theorem 5.

```
(A \simeq B) \simeq \mathsf{Perm}|A|
```

A Calculus of Permutations. Syntactic theories only rely on transforming source programs to other programs, much like algebraic calculation. Since only the *syntax* of the programming language is relevant to the syntactic theory, the theory is accessible to non-specialists like programmers or students.

In more detail, it is a general problem that, despite its fundamental value, formal semantics of programming languages is generally inaccessible to the computing public. As Schmidt argues in a recent position statement on strategic directions for research on programming languages [?]:

...formal semantics has fed upon increasing complexity of concepts and notation at the expense of calculational clarity. A newcomer to the area is expected to specialize in one or more of domain theory, inuitionistic type theory, category theory, linear logic, process algebra, continuation-passing style, or whatever. These specializations have generated more experts but fewer general users.

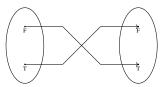
A Calculus of Permutations. First conclusion: it might be useful to *reify* a (sound and complete) set of equivalences as combinators, such as the fundamental "proof rules" of semirings:

```
\begin{array}{lll} \operatorname{data} & \longleftrightarrow : \mathsf{U} \to \mathsf{U} \to \mathsf{Set} \text{ where} \\ & \operatorname{unite}_+ & : (t \colon \mathsf{U}) \to \mathsf{PLUS} \mathsf{ZERO} \ t \longleftrightarrow t \\ & \operatorname{uniti}_+ & : (t \colon \mathsf{U}) \to \mathsf{PLUS} \mathsf{ZERO} \ t \longleftrightarrow t \\ & \operatorname{uniti}_+ & : (t \colon \mathsf{U}) \to \mathsf{PLUS} \mathsf{ZERO} \ t \\ & \operatorname{swap}_+ & : (t_1 \ t_2 \colon \mathsf{U}) \to \mathsf{PLUS} \ t_1 \ t_2 \longleftrightarrow \mathsf{PLUS} \ t_2 \ t_1 \\ & \operatorname{assocl}_+ : (t_1 \ t_2 \ t_3 \colon \mathsf{U}) \to \mathsf{PLUS} \ t_1 \ \mathsf{PLUS} \ t_2 \ t_3) \longleftrightarrow \mathsf{PLUS} \ (\mathsf{PLUS} \ t_1 \ t_2) \ t_3 \\ & \operatorname{assocr}_+ : (t_1 \ t_2 \ t_3 \colon \mathsf{U}) \to \mathsf{PLUS} \ \mathsf{PLUS} \ t_1 \ t_2) \ t_3 \longleftrightarrow \mathsf{PLUS} \ t_1 \ (\mathsf{PLUS} \ t_2 \ t_3) \\ & \operatorname{unite}_+ & : (t \colon \mathsf{U}) \to \mathsf{TIMES} \ \mathsf{ONE} \ t \longleftrightarrow \mathsf{Uniti}_+ \\ & : (t \colon \mathsf{U}) \to \mathsf{TIMES} \ \mathsf{ONE} \ t \longleftrightarrow \mathsf{Uniti}_+ \\ & : (t_1 \ t_2 \ t_3 \colon \mathsf{U}) \to \mathsf{TIMES} \ t_1 \ t_2 \longleftrightarrow \mathsf{TIMES} \ t_2 \ t_3 \\ & : \mathsf{assocl}_+ : (t_1 \ t_2 \ t_3 \colon \mathsf{U}) \to \mathsf{TIMES} \ t_1 \ \mathsf{UNES} \ t_2 \ t_3 \longleftrightarrow \mathsf{TIMES} \ t_1 \ \mathsf{UNES} \ t_2 \ t_3 \\ & : \mathsf{assocl}_+ : (t_1 \ t_2 \ t_3 \colon \mathsf{U}) \to \mathsf{TIMES} \ \mathsf{UNES} \ \mathsf{UNES} \ t_1 \ t_2 \ t_3 \\ & : \mathsf{UNES} \ t_1 \ t_2 \ t_3 : \mathsf{U} \to \mathsf{UNES} \ \mathsf{UNES} \ \mathsf{UNES} \ t_1 \ \mathsf{UNES} \ t_2 \ \mathsf{UNES} \ \mathsf{UNE
```

```
\begin{array}{lll} \textbf{absorbr} & : (t : \textbf{U}) \rightarrow \textbf{TIMES ZERO } t \longleftrightarrow \textbf{ZERO} \\ \textbf{absorbl} : (t : \textbf{U}) \rightarrow \textbf{TIMES } t \textbf{ZERO} \longleftrightarrow \textbf{ZERO} \\ \textbf{factorzr} : (t : \textbf{U}) \rightarrow \textbf{ZERO} \longleftrightarrow \textbf{TIMES } t \textbf{ZERO} \\ \textbf{factorzl} : (t : \textbf{U}) \rightarrow \textbf{ZERO} \longleftrightarrow \textbf{TIMES } t \textbf{ZERO} \\ \textbf{factorzl} : (t : \textbf{U}) \rightarrow \textbf{ZERO} \longleftrightarrow \textbf{TIMES } t \textbf{ZERO} \\ \textbf{dist} & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow \textbf{TIMES} (\textbf{PLUS} t_1 t_2) t_3 \longleftrightarrow \textbf{PLUS} (\textbf{TIMES} t_1 t_3) (\textbf{TIMES} t_2 t_3) \\ \textbf{factor} & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow \textbf{PLUS} (\textbf{TIMES} t_1 t_3) (\textbf{TIMES} t_2 t_3) \longleftrightarrow \textbf{TIMES} (\textbf{PLUS} t_1 t_2) t_3 \\ \textbf{id} \longleftrightarrow & : (t : \textbf{U}) \rightarrow t \longleftrightarrow t \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow \textbf{U} (\textbf{TIMES} t_1 t_2) \rightarrow (t_2 \longleftrightarrow t_3) \rightarrow (t_1 \longleftrightarrow t_3) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{PLUS} t_1 t_2 \longleftrightarrow \textbf{PLUS} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & = \bigcirc \quad & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_1 t_2 \longleftrightarrow \textbf{TIMES} t_3 t_4) \\ & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{TIMES} t_3 t_4) \\ & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{U}) \\ & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow (t_2 \longleftrightarrow t_4) \rightarrow (\textbf{U}) \\ & : (t_1 t_2 t_3 : \textbf{U}) \rightarrow (t_1 \longleftrightarrow t_3) \rightarrow
```

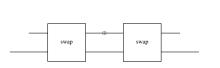
# 3. Example Circuit: Simple Negation





```
BOOL: U
BOOL = PLUS ONE ONE
n_1: BOOL \longleftrightarrow BOOL
n_1 = swap_{\perp}
```

Example Circuit: Not So Simple Negation.





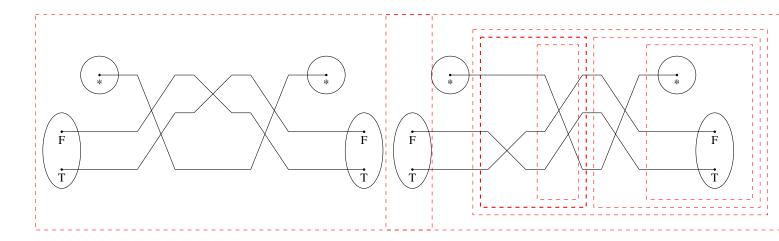
```
\begin{array}{ll} \mathbf{n}_2: \mathbf{BOOL} \longleftrightarrow \mathbf{BOOL} \\ \mathbf{n}_2 = & \mathbf{uniti} \star \odot \\ & \mathbf{swap} \star \odot \\ & (\mathbf{swap}_+ \otimes \mathbf{id} \longleftrightarrow) \odot \\ & \mathbf{swap} \star \odot \\ & \mathbf{unite} \star \end{array}
```

Reasoning about Example Circuits. Algebraic manipulation of one circuit to the other:

```
\mathsf{negEx} = \mathsf{uniti} \star \overset{\cdot}{\odot} (\mathsf{swap} \star \odot ((\mathsf{swap}_+ \otimes \mathsf{id} \longleftrightarrow) \odot (\mathsf{swap} \star \odot \mathsf{unite} \star)))
                            \Leftrightarrow \langle id \Leftrightarrow \Box assoc \odot I \rangle
              \mathsf{uniti} \star \ \odot \ ((\mathsf{swap} \star \ \odot \ (\mathsf{swap}_+ \ \otimes \ \mathsf{id} \longleftrightarrow)) \ \odot \ (\mathsf{swap} \star \ \odot \ \mathsf{unite} \star))
                            \Leftrightarrow \langle \; \mathsf{id} \Leftrightarrow \; \boxdot \; (\mathsf{swapl} \, \star \, \Leftrightarrow \; \boxdot \; \mathsf{id} \Leftrightarrow) \; \rangle
              \mathsf{uniti} \star \odot (((\mathsf{id} \longleftrightarrow \otimes \mathsf{swap}_+) \odot \mathsf{swap} \star) \odot (\mathsf{swap} \star \odot \mathsf{unite} \star))
                             ⇔⟨id⇔ ⊡ assoc⊙r⟩
             \mathsf{uniti}_{\star} \odot ((\mathsf{id} \longleftrightarrow \otimes \mathsf{swap}_{+}) \odot (\mathsf{swap}_{\star} \odot (\mathsf{swap}_{\star} \odot \mathsf{unite}_{\star})))
                              \Leftrightarrow \langle id \Leftrightarrow \boxdot (id \Leftrightarrow \boxdot assoc \odot I) \rangle
            \begin{array}{l} \mathsf{uniti} \star \odot ((\mathsf{id} \longleftrightarrow \otimes \mathsf{swap}_+) \odot ((\mathsf{swap} \star \odot \mathsf{swap} \star) \odot \mathsf{unite} \star)) \\ \Leftrightarrow \langle \mathsf{id} \Leftrightarrow \boxdot (\mathsf{id} \Leftrightarrow \boxdot (\mathsf{linv} \odot \mathsf{l} \boxdot \mathsf{id} \Leftrightarrow)) \rangle \end{array}
              uniti∗ ⊙ ((id←
                           i \star \odot ((id \longleftrightarrow \otimes swap_+) \odot (id \longleftrightarrow \odot unite \star))
\Leftrightarrow \langle id \Leftrightarrow \boxdot (id \Leftrightarrow \boxdot idl \odot l) \rangle
              uniti∗ ⊙ ((id←
                                                                      \otimes swap_+) \odot unite\star)
                            \Leftrightarrow \langle \; \mathsf{assoc} \odot \mathsf{I} \; \rangle
             (uniti* \odot (id \longleftrightarrow \otimes swap_{+})) \odot unite*
                            \Leftrightarrow \langle \, \mathsf{unitil} \, \star \Leftrightarrow \, \boxdot \, \mathsf{id} \, \Leftrightarrow \,
            (swap<sub>+</sub> ⊙ uniti∗) ⊙ unite∗
⇔⟨assoc⊙r⟩
              swap_{+} \odot (uniti \star \odot unite \star)
\Leftrightarrow \langle id \Leftrightarrow \boxdot linv \odot l \rangle
                        ap<sub>+</sub> ⊙ id←
                            ⇔ (idr⊙l)
```

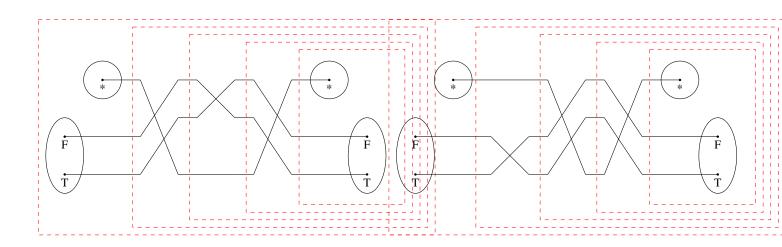
Visually.
Original circuit:

2 2015/6/13



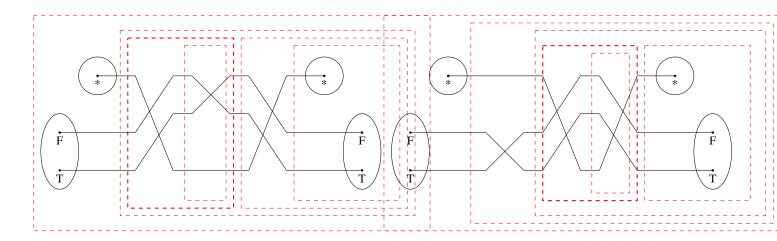
Making grouping explicit:

By associativity:



By associativity:

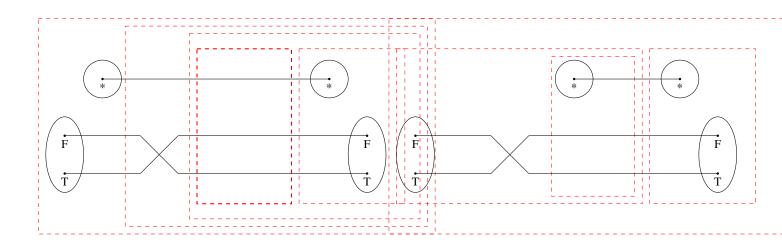
By associativity:



By pre-post-swap:

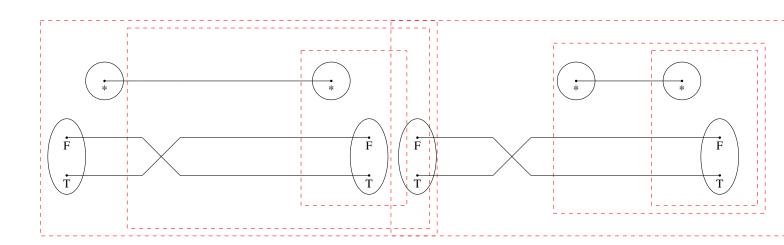
By swap-swap:

3 2015/6/13



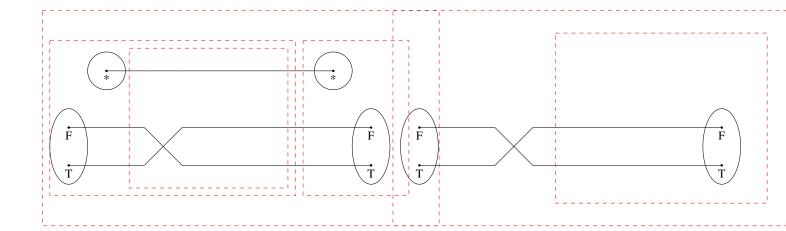
By id-compose-left:

By associativity:



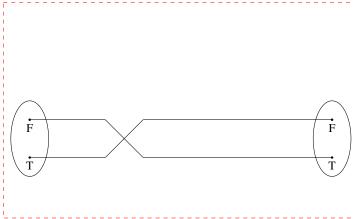
By associativity:

By unit-unit:



By swap-unit:

By id-unit-right:



Categorification II. The categorification of a semiring is called a Rig Category. As with a semiring, there are two monoidal structures, which interact through some distributivity laws.

**Theorem 6.** The following are Symmetric Bimonoidal Groupoids:

- The class of all types (Set)
- The set of all finite types
- The set of permutations
- The set of equivalences between finite types
- Our syntactic combinators

The coherence rules for Symmetric Bimonoidal groupoids give us 58 rules.

Categorification III.

Conjecture 1. The following are Symmetric Rig Groupoids:

- The class of all types (Set)
- The set of all finite types, of permutations, of equivalences between finite types
- Our syntactic combinators

coherence rules for Symmetric Rig Categories.

Conjecture 2. The set of coherence rules for Symmetric Rig *Groupoids are a sound and complete set for circuit equivalence.* 

# 4. But is this a programming language?

We get forward and backward evaluators  $eval: \{t_1 \ t_2 : \mathbf{U}\} \to (t_1 \longleftrightarrow t_2) \to [t_1] \to [and] \text{ of course the punchline:}$ which really do behave as expected which really do behave as expected calculus:  $[t_1 \ t_2 : U) \to (c:t_1 \longleftrightarrow t_2) \to [$  Theorem 7 (Laplaza 1972). There is a sound and complete set of Manipulating circuits. Nice framework, but:

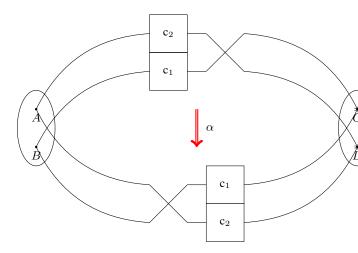
- We don't want ad hoc rewriting rules.
  - Our current set has 76 rules!
- Notions of soundness; completeness; canonicity in some sense.
  - Are all the rules valid? (yes)
  - Are they enough? (next topic)
  - Are there canonical representations of circuits? (open)

## 5. Categorification I

Type equivalences (such as between  $A \times B$  and  $B \times A$ ) are

Equivalences between Functors are Natural Isomorphisms. At the value-level, they induce 2-morphisms:

```
\begin{array}{l} \mathbf{c}_1: \{B\ C: \mathbf{U}\} \to B \longleftrightarrow C \\ \mathbf{c}_2: \{A\ D: \mathbf{U}\} \to A \longleftrightarrow D \end{array}
       \mathbf{p_1} \ \mathbf{p_2} : \{A \ B \ C \ D : \mathbf{U}\} \rightarrow \mathbf{PLUS} \ A \ B \longleftrightarrow \mathbf{PLUS} \ C \ D
        \mathbf{p}_2 = (\mathbf{c}_2 \oplus \mathbf{c}_1) \odot \mathbf{swap}_{\perp}
2-morphism of circuits
```



#### 6. Emails

Reminder of

http://mathoverflow.net/questions/106070/int-constructi

Also.

5

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1 seems relevant

Indeed, this does not seem to be in the library.

On 2015-04-10 10:52 AM, Amr Sabry wrote: I had checked and found no traced categories or Int con

The story without trace and without the Int construction

On 04/10/2015 09:06 AM, Jacques Carette wrote: I don't know, that a "symmetric rig" (never mind higher programming language, even if only for "straight line p interesting! ;)

But it really does depend on the venue you'd like to se POPL, then I agree, we need the Int construction. can be made, the better.

It might be in 'categories' already! Have you looked?

In the meantime, I will try to finish the Rig part. Th conditions are non-trivial. Jacques

On 2015-04-10, 06:06 , Sabry, Amr A. wrote: I am thinking that our story can only be compelling if that h.o. functions might work. We can make that case b implementing the Int Construction and showing that a li h.o. functions emerges and leave the big open problem of the multiplication etc. for later work. I can start wor will require adding traced categories and then a generi

```
Unfortunately not. Yes, there is a general feeli Tigeostore awethers, thate Iardawithout itheownt construction
I do believe that all our terms have computationaOnrO46$0/301$re0@a06tAMetJatques Carette wrote:
                                                 I don't know, that a "symmetric rig" (never mind higher
Note that at level 1, we have equivalences betweeprசுதுகோடிக்கு AadgRage (As Ben instoReymf6rD)stitookhatline ந
                                                 interesting! ;)
Yes, we should dig into the Licata/Harper work and adapt to our setting.
                                                 But it really does depend on the venue you'd like to se
Though I think we have some short-term work that DePsimphenneedgteedowtoneedutbeountwookswiddtiest ofine
                                                 can be made, the better.
Jacques
                                                 It might be in 'categories' already! Have you looked?
On 2015-04-09 12:05 PM, Amr Sabry wrote:
Trying to get a handle on what we can transport of nmbire percuismedy Ifwielcanytransportshthing Righpar HoTTTh
                                                 conditions are non-trivial.
(I use permutation for level 0 to avoid too many useques 'equivalence' which gets confusing.)
Level 0: Given two types A and B, if we have a pe@mu2@15004be@we@6:@Kem ShbnyweAmmnAtrwnspert something
                                                 I am thinking that our story can only be compelling if
For example: take P = . + C; we can build a permuthatioh.betweentAbcsamdgBtCwGrbm Whecqnivmakpermatatase k
                                                 implementing the Int Construction and showing that a li
                                                 h.o. functions emerges and leave the big open problem of
                                                 the multiplication etc. for later work. I can start wor
Level 1: Given types A, B, C, and D. let Perm (A, B) ibe theutypeaddipgrimmtaeddonatheoween Andnthenaadgenemi
                                                 Construction in the categories library. What do you thi
This is more interesting. What's a good example though of a property P that we can implement?
                                                 On Apr 9, 2015, at 10:59 PM, Jacques Carette <carette@m
In think that in HoTT the only way to do this trawspost is via univalence. First you find an equivalence
In HoTT this is exhibited by the failure of canonichave thesedateims, that aremetnick.sweucanfesgeonelosec
                                                 RigCategory as well, but very close.
Perhaps we can adapt the discussion/example in http://homotopytypetheory.org/2011/07/27/canonicity-for-2
                                                 Of course, we're still missing the coherence conditions
--Amr
                                                 Jacques
I hope not! [only partly joking]
                                                 solutions to quintic equations proof by arnold is all a
Actually, there is a fair bit about this that I dislike: it seems to over-simplify by arbitrarily saying
                                                 I thought we'd gotten at least one version, but could n
On 2015-04-09 12:36 PM, Amr Sabry wrote:
This came up in a different context but looks likeni20h5ght-b5 &seTuAMtoSabrtpoAmr A. wrote:
                                                 Didn't we get stuck in the reverse direction. We never
http://arxiv.org/pdf/gr-qc/9905020
                                                 On Apr 25, 2015, at 8:27 AM, Jacques Carette <carette@m
Separate. The Grothendieck construction in this case is about fibrations, and is not actually related t
                                                 Right. We have one direction, from Pi combinators to F
Jacques
                                                 Note that quite a bit of the code has (already!!) bit-r
                                               6
```

Construction in the categories library. What do you 2015k04-1Amr1:56 AM, Sabry, Amr A. wrote:

seems relevant

https://pigworker.wordpress.com/2015/04/01/warming-hapdtohdokedopandtfpendheoryfaced categories or Int con

On Apr 9, 2015, at 10:59 PM, Jacques Carette <carette@mcmaster.ca>

I have the braiding, and symmetric structures don  ${\tt RemiMdst}$  of the

Of course, we're still missing the coherence condAtions for Rig.

RigCategory as well, but very close.

On 2015-04-09 11:41 AM, Sabry, Amr A. wrote:

Can you make sense of how this relates to us?

wrote:

Jacques

Yes. The categories library has a Grothendieck construct

On Apr 10, 2015, at 11:04 AM, Jacques Carette <caretted

http://mathoverflow.net/questions/106070/int-constructi

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1

Indeed, this does not seem to be in the library.

On 2015-04-10 10:52 AM, Amr Sabry wrote:

A  $\simeq$  B if exists f : A  $\rightarrow$  B such that:

We do not have the other direction currently in the (excitets ThatBmay Anoxithe grown foad, ichas) whe do have LeftCa (exists  $h : B \rightarrow A$  with f o  $h \sim idB$ )

Jacques

On 2015-04-25 7:28 AM, Sabry, Amr A. wrote: That's obsolete for now.

Does this definition reduce to our semantic notion of p and B are finite sets?

--Amr

By the way, do we have a complement to thm2 that connects to Pi. Ideally what we want to say is what I s

On Apr 21, 2015, at 11:03 AM, Jacques Carette <carette@ On Apr 24, 2015, at 5:25 PM, Jacques Carette <caretteencmaster.ca> wrote:

Jacques

Is that going somewhere, or is it an experiment than showlthbe pattibles potencerned that our code do match that. But since we have no specific deadline, I bit more time isn't too bad.

Thanks. I like that idea ;).

Since propositional equivalence is really HoTT equivale I have a bunch of things I need to do, so I won'tIremalnotputoomgoheedneedneedtaboohtisthatisideeoweekends -- our permutations should be the same whether in HoTT or in r

I understand the desire to not want to rely on theifhliarbbesenceicosdofieqsivaleadep depéciahbw sowceom code was lifted from a previous HoTT-based attempt at t As I was trying really hard to come up with a single story, I am a little confused as to what "my" story

I would certainly agree with the not-not-statement: usi On 2015-04-23 9:07 PM, Sabry, Amr A. wrote: equivalence known to be incompatible with HoTT is not a Instead of discussing this over and over, I think it is clear that thm2 will be an important part of any

Jacques On Apr 23, 2015, at 6:07 PM, Amr Sabry <sabry@indiana.edu> wrote:

On 2015-04-21 10:38 AM, Sabry, Amr A. wrote:

I wasn't too worried about the symmetric vs. non-SymmetrichabtioshouleqsiantebreingheoHwffteodawhas mare story so that we can see how things fit together. I am I do recall the other discussion about extensionafiowardshatHdTSenssabedcsnckydwhiwhthstwhatid&asthatedheI

--Amr

11th, and Dan Licata's reply?

we should have a different initial bias let me know. I just really want to avoid the full reliance on the coherence conditions. I also noted you have a diffe What is there is just one paragraph for now but it alre question: if we are pursuing that HoTT story we should prove that the HoTT notion of equivalence when speciali On 04/23/2015 12:23 PM, Jacques Carette wrote: types reduces to permutations. That should be a strong Did you see my "HoTT-agda" question on the Agda mahidingstishdothMapracise notion of permutation we get ( by enumerations or not should help quite a bit).

What you wrote reduces to our definition of \*equiMalengenerably always keeping our notions of equivalence permutation. To prove that equivalence, we wouldlevedsftoextin-space mith the HoTT definitions seems to question of February 18th on the Agda mailing listhing to do. --Amr

Another way to think about it is that this is EXACTLYawdafftthm@se coherence conditions are really comple provides: a proof that for finite A and B, equivalence between A and B (as below) is equivalent to permutations implementedtassufferp, we exculpt get a nice language for expressing pf).

--Amr

Now, we may want another representation of permutations which uses functions (qua bijections) internally instead of Opc@0f27/20henO6he6 AM, Sabry, Amr A. wrote: answer to your question would be "yes", modulo thHequessianniaoswedeabowe need a canonical form for every which encoding of equivalence to use.

Indeed! Good idea.

Jacques

However, it may not give us a normal form. This is bed

On 2015-04-23 10:32 AM, Sabry, Amr A. wrote: Thought a bit more about this. We need a little bfidgeh frow o HdST because we have associativity and commu our code and we're good to go I think.

However, I think it is not that bad: we can use the obj In HoTT we have several notions of equivalence that are equivalent (in

the technical sense). The one that seems easiest Herwork whother theught:

1. think of the combinators as polynomials in 3 operators following: 2. expand things out, with + being outer, \* middle, . i

7

```
On 2015-06-02 7:53 PM, Sabry, Amr A. wrote:
the issue is that the re-ordering could produce newokingdatorthate2pasth Buctwith.a welthesefwedetphysocal
                                                 There are some slightly different approaches to impleme
Jacques
On 2015-04-27 6:16 AM, Sabry, Amr A. wrote:
                                                A category can be formalized as a kind of elementary ax
Here is a nice idea: we need a canonical form for every pi-combinator. Our previous approach gave us some
                                                                 f:X to Y equiv Domain(f) = X and Range(
I've been thinking about this some more. I can't help but think that, somehow, Laplaza has already work
                                                 is used for the three place predicate.
Pi-combinators might be simpler, I don't know.
                                                 The operations such as the binary composition of maps a
Another place to look is in Fiore (et al?)'s proof of completeness of a similar case. Again, in their of
                                                 f:Z to Y, g:Y to X implies g(f):Z to X
On 2015-04-26 6:34 AM, Sabry, Amr A. wrote:
What's the proof strategy for establishing that aACfementiemplsymbalPthatombiwatorproducesigiman wideh wasni
Well enough. Last talk on the last day, so peoplEoarmostired.thDospstwmesvehatuBeddawerewookedion tiheraweom
I think the idea that (reversible circuits == proAfmtephis misfjantamidnitset tsongidon foructors sink fund
If we had a similar story for Caley+T (as they liWethotbaslrep) esentation bave madegar bigsemgsplasms in
Note that I've pushed quite a few things forward im whitipodeyouMosslame quitehstmanightfoiward, obstetwingt
Yes, I think this can make a full paper -- especiWelyrenin wemfinenkethotegonifyingrese nātidapefidscamhi
I think the details are fine. A little bit of polibelingvestperbabayaadanohatas defididateloforSome oftege
Writing it up actually forced me to add PiEquiv.annastostheerapostequyy-ofwhymhetsicrmivnaldahowatehories
Firstly, thanks Spencer for setting this up. In any symmetric monoidal 2-category, we have a notion
This is partly a response to Amr, and partly my owhotankesonC (@computing, with) graphicatclanguages for mor
One of the key ingredients to getting diagrammaticnlaguagesItwodbdwooksfderybhiasisoaareaadhybtakdesbei
If you ignore these theorems and insist on workinkjowidhthine typeaxfothinegosidalsfategoripsrpoatherothaned
Of course, when it comes to computing with diagramsitehee Faired thdeed you Bhaveuth make potebose it seems!
                                                 Jacques
(1: combinatoric) its a graph with some extra bells and whistles
(2: syntactic) its a convenient way of writing down 2016-Q5-nd7o8:CerAM, Sabry, Amr A. wrote:
(3: "lego" style) its a collection of tiles, connactedhingethesennta & Drpwank http://www.informatik.uni-
Point of view (1) is basically what Quantomatic is-Ammilt on. "String graphs" aka "open-graphs" give a co
Naiively, point of view (2) is that a diagram repMesentslabedqwdvkleaseIceassunteeschonsutinlabersynt
Point of view (3) is the one espoused by the 2D/hDighgrandinewsiting and peoples (bagont ves Lafont ar
                                                 http://iml.univ-mrs.fr/~lafont/pub/diagrams.pdf
This eliminates the need for the interchange law, but keeps pretty much everything else "rigid". This be
                                                 A Homotopical Completion Procedure with Applications to
This is a very good example of CCT. As I am sure http://odraps.datetrshondehepuisefraet.goorDphpasoRoss_okus
My primary CCT interest, so far, has been with whatreaballnioempsetatofnalidepotest Thissisateslighh sfreh
                                                 http://www.lix.polytechnique.fr/Labo/Samuel.Mimram/docs
There's also the perspective that string diagrams of various flavors are morphisms in some operad (the
                                                 I think there is something very important going on in s
From that perspective, the string diagrams for trated: Monomidan quadegour lesgaree Mlabers of by that bije
                                                 which I also attach. [I googled 'Knuth Bendix coherence
Yes, I am sure this observation has been made before. We'd have to verify it for all the 2-paths before
```

8

3. within each . term, use combinators to re-ordefAhddisigsceweowoidalneadeqorpeskazecanvalved omdkmotforhe

4. show this terminates

2015/6/13

There are also seems to be relevant stuff buried (very

Also, Tarmo Uustalu's "Coherence for skew-monoidal [Apparently I could have saved myself some of that Somehow, at the end of the day, it seems we're look