Title: Physics, Topology, Logic, and Computation: A Rosetta Stone

Date: Jun 03, 2009 12:15 PM

URL: http://pirsa.org/09060022

Abstract: TBA

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

The Rosetta Stone (pocket version)

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

The Rosetta Stone (pocket version)

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Objects

- String diagrams have 'strings' or 'wires':

X 

- Quantum mechanics has Hilbert spaces: $X \cong \mathbb{C}^n$
- Topology has manifolds: X 
- Linear logic has propositions:

X = "I have an item of type X."

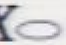
- Computation has datatypes: `interface X;`
- SET has sets: X

3

Objects

- **String diagrams have 'strings' or 'wires':**

X 

- **Quantum mechanics has Hilbert spaces:** $X \cong \mathbb{C}^n$
- **Topology has manifolds:** X 
- **Linear logic has propositions:**

X = **"I have an item of type** X**."**

- **Computation has datatypes:** `interface X;`
- **SET has sets:** X

3

Morphisms

- **String diagrams have vertices:**

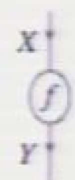
- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **String diagrams have vertices:**



- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **String diagrams have vertices:**

- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **String diagrams have vertices:**

- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

4

Morphisms

- **String diagrams have vertices:**

- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- String diagrams have vertices:

- Quantum mechanics has linear transformations:

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **String diagrams have vertices:**

- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **Topology has cobordisms:** X  Y
- **Linear logic has constructive proofs:**

$$X \vdash Y$$

- **Computation has (roughly) programs:** `Y f(X);`
- **SET has functions:** $f : X \to Y$

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

The Rosetta Stone (pocket version)

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

The Rosetta Stone (pocket version)

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

The Rosetta Stone (pocket version)

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Objects

- String diagrams have ‘strings’ or ‘wires’:

X 

- Quantum mechanics has Hilbert spaces: $X \cong \mathbb{C}^n$
- Topology has manifolds: X 
- Linear logic has propositions:

X = “I have an item of type X.”

- Computation has datatypes: `interface X;`
- SET has sets: X

Morphisms

- **String diagrams have vertices:**

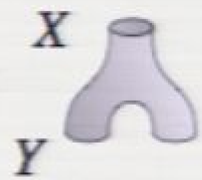
- **Quantum mechanics has linear transformations:**

$$f : X \to Y \quad \cong \quad f : \mathbb{C}^n \to \mathbb{C}^m$$

(An $m \times n$ matrix with complex entries)

Morphisms

- **Topology has cobordisms:**



- **Linear logic has constructive proofs:**

$$X \vdash Y$$

- **Computation has (roughly) programs:** `Y f(X);`
- **SET has functions:** $f : X \to Y$

5

Morphisms compose associatively

- **String diagrams:**

- **Quantum mechanics: matrix multiplication**
- **Topology:**

6

Morphisms compose associatively

- **Linear logic:** $\dfrac{Y \vdash Z \quad X \vdash Y}{X \vdash Z}\ (\circ)$

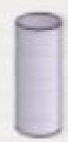
- **Computation:**

```
Y f(X x);
Z g(Y y);
...
z = g(f(x));
```

- **SET:** $(g \circ f) : X \to Z$

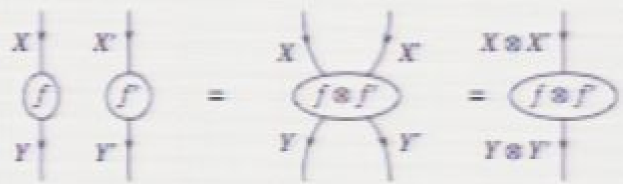
Identity morphisms

- **String diagrams:** 

- **Quantum mechanics: identity matrix** $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- **Topology:** 
- **Linear logic:** $\dfrac{}{X \vdash X}\,(\mathrm{i})$
- **Computation:** `X id(X x) { return x; }`
- **SET:** $1_X : X \to X$

Monoidal categories

- **String diagrams:**



- **Quantum mechanics: tensor product**

$$\left(\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right) \otimes \begin{pmatrix} e & f & g \\ h & j & k \end{pmatrix} = \left(\begin{array}{ccc|ccc} ae & af & ag & be & bf & bg \\ ah & aj & ak & bh & bj & bk \\ \hline ce & cf & cg & de & df & dg \\ ch & cj & ck & dh & dj & dk \end{array}\right)$$

9

Monoidal categories

- **Topology:** 

- **Linear logic: AND** $\dfrac{X \vdash Y \quad X' \vdash Y'}{X \otimes X' \vdash Y \otimes Y'}\ (\otimes)$

- **Computation: parallel programming**

```
ParallelPair<X, Xp> pair;
```

- **SET:** $f \times f' : X \times X' \to Y \times Y'$

10

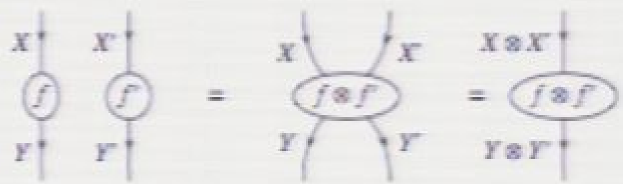
Identity morphisms

- **String diagrams:** 

- **Quantum mechanics: identity matrix** $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- **Topology:** 
- **Linear logic:** $\overline{ X \vdash X }$ (i)
- **Computation:** `X id(X x) { return x; }`
- **SET:** $1_X : X \to X$

Monoidal categories

- **String diagrams:**



- **Quantum mechanics: tensor product**

$$\left(\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right) \otimes \begin{pmatrix} e & f & g \\ h & j & k \end{pmatrix} = \left(\begin{array}{ccc|ccc} ae & af & ag & be & bf & bg \\ ah & aj & ak & bh & bj & bk \\ \hline ce & cf & cg & de & df & dg \\ ch & cj & ck & dh & dj & dk \end{array}\right)$$

9

Monoidal categories

- **Topology:** 
- **Linear logic: AND** $\dfrac{X \vdash Y \quad X' \vdash Y'}{X \otimes X' \vdash Y \otimes Y'}\ (\otimes)$
- **Computation: parallel programming**

```
ParallelPair<X, Xp> pair;
```

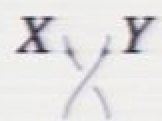
- **SET:** $f \times f' : X \times X' \to Y \times Y'$

Monoidal unit

- **String diagram:**
- **Quantum mechanics:** $I = \mathbb{C}$**, the phase of a photon**
- **Topology:**
- **Linear logic:** I**, trivial proposition**
- **Computation:** `I = void` **or** `I = unit` **type**
- **SET: one-element set** I

Braided monoidal categories

- **String diagrams:**



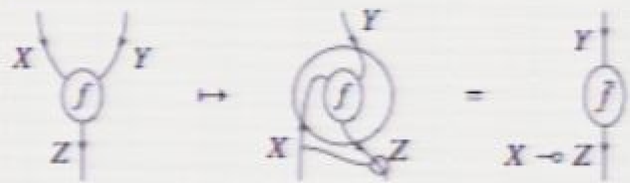
- **Quantum mechanics: swap the particles. Bosons commute, fermions anticommute; quantized magnetic flux tubes in thin films, or "anyons", can have arbitrary phase multiplier.**

Braided monoidal categories

- **Topology:** 
- **Linear logic:** $\dfrac{W \vdash X \otimes Y}{W \vdash Y \otimes X}$ (b)
- **Computation:** `pair.swap();`
- **SET:** $b(\langle x, y \rangle) = \langle y, x \rangle$

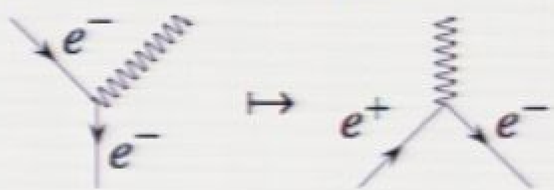
Braided monoidal closed categories

- **String diagrams:**



- **Quantum mechanics: antiparticles**

$$r_X : X \otimes I \to X \quad \cong \quad \text{pair} : I \to X^* \otimes X$$



Braided monoidal closed categories

- **Topology:** 

- **Linear logic: IMPLIES** $\dfrac{X \otimes Y \vdash Z}{Y \vdash X \multimap Z}$ (c)
- **Computation: Currying**

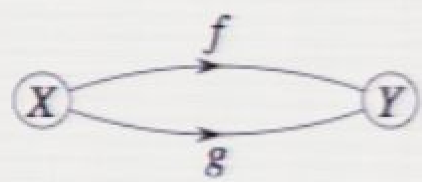
```
z = f(x, y);
```

or

```
z = f(y)(x);
```

- **SET:** $f : X \times Y \rightarrow Z \quad \cong \quad f : Y \rightarrow Z^X$

Model Theory



Braided monoidal closed categories

- **Topology:** 
- **Linear logic: IMPLIES** $\dfrac{X \otimes Y \vdash Z}{Y \vdash X \multimap Z}$ (c)
- **Computation: Currying**

```
z = f(x, y);
```

or

```
z = f(y)(x);
```

- **SET:** $f : X \times Y \to Z \quad \cong \quad f : Y \to Z^X$

15

Braided monoidal closed categories

- **Topology:**

- **Linear logic: IMPLIES** $\dfrac{X \otimes Y \vdash Z}{Y \vdash X \multimap Z}$ (c)
- **Computation: Currying**

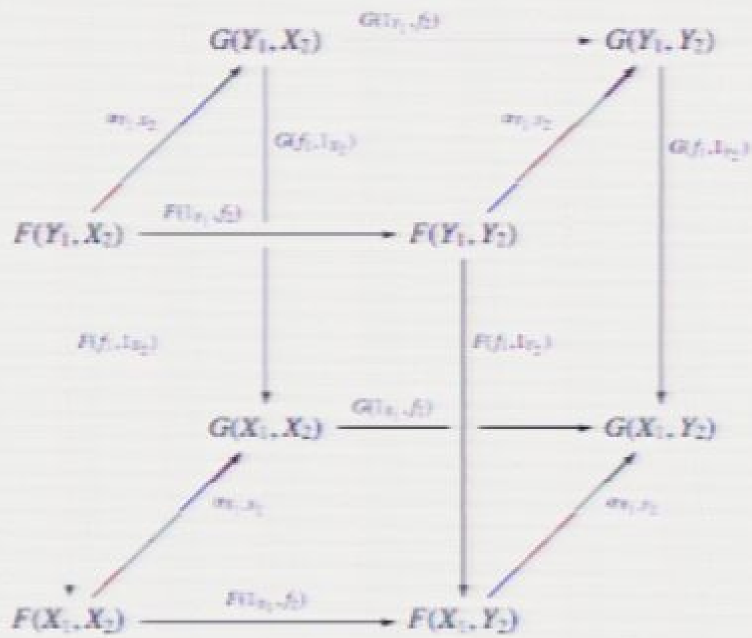
```
z = f(x, y);
    or
z = f(y)(x);
```

- **SET:** $f : X \times Y \to Z \quad \cong \quad f : Y \to Z^X$

Preview File Edit View Go Tools Bookmarks Window Help (3:30) Wed 12:38 PM

rosetta.pdf (page 48 of 73)

Previous Next Zoom Move Text Select Sidebar Search



If $C_1 = C_2$, we can choose a single object X and a single morphism $f\colon X \to Y$ and use it in both slots. As shown in Figure [] there are then two paths from one corner of the cube to the antipodal corner that only involve α for repeated arguments: that is, $\alpha_{X,X}$ and $\alpha_{Y,Y}$, but not $\alpha_{X,Y}$ or $\alpha_{Y,X}$. These paths give a commuting hexagon.

This motivates the following:

Definition 22 A **dinatural transformation** *$\alpha\colon F \Rightarrow G$ between functors $F, G\colon C^{op} \times C \to D$ assigns to every object X in C a morphism $\alpha_X\colon F(X,X) \to G(X,X)$ in D such that for every morphism $f\colon X \to Y$ in C, the hexagon in Figure [] commutes.*

In the case of the identity rule, this commuting hexagon follows from the fact that the identity morphism is a

Preview File Edit View Go Tools Bookmarks Window Help (3:20) Wed 12:39 PM

rosetta.pdf (page 49 of 73)

Previous Next Zoom Move Text Select Sidebar Search

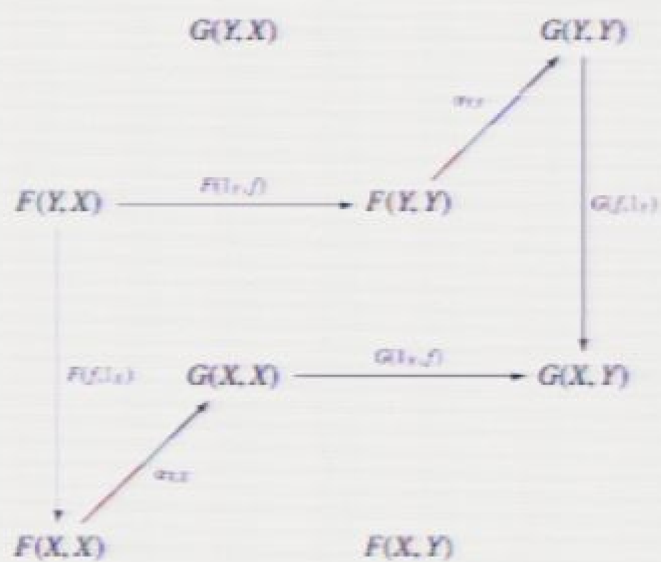


Figure 1: A natural transformation between functors $F, G: C^{op} \times C \to D$ gives a commuting cube in D for any morphism $f: X \to Y$, and there are two paths around the cube that only involve α for repeated arguments.

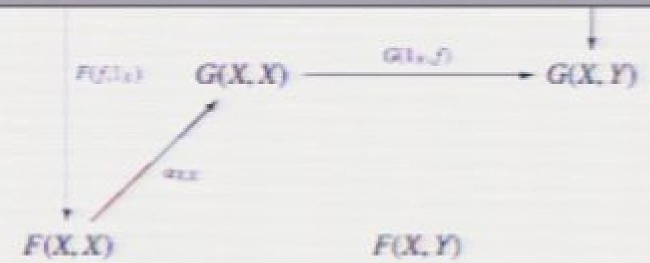


Figure 1: A natural transformation between functors $F, G: C^{op} \times C \to D$ gives a commuting cube in D for any morphism $f: X \to Y$, and there are two paths around the cube that only involve α for repeated arguments.

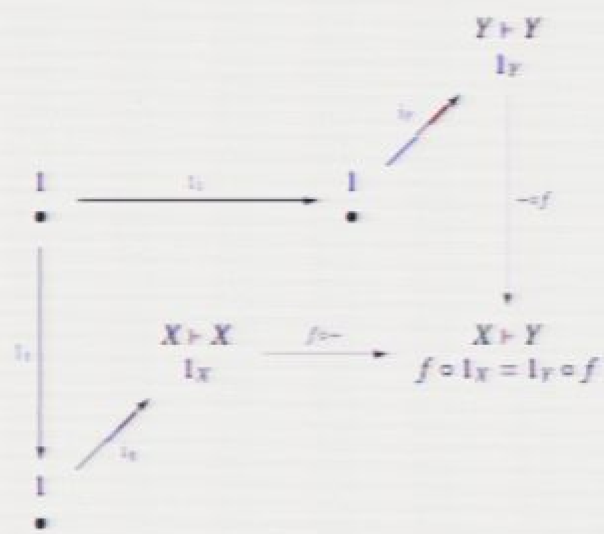


Figure 2: Dinaturality of the (i) rule, where $f: X \to Y$. Here $\bullet \in 1$ denotes the one element of the one-element set.

Figure 1: A natural transformation between functors $F, G: C^{op} \times C \to D$ gives a commuting cube in D for any morphism $f: X \to Y$, and there are two paths around the cube that only involve α for repeated arguments.

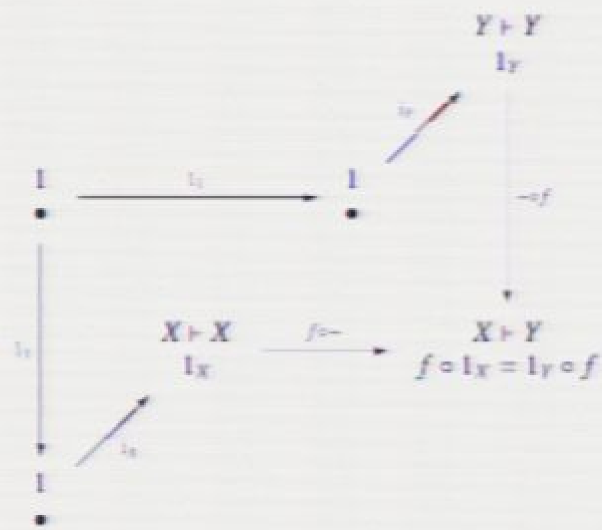


Figure 2: Dinaturality of the (i) rule, where $f: X \to Y$. Here $\bullet \in 1$ denotes the one element of the one-element set.

Preview File Edit View Go Tools Bookmarks Window Help (3:13) Wed 12:41 PM

rosetta.pdf (page 50 of 73)

Previous Next Zoom Move Text Select Sidebar Search

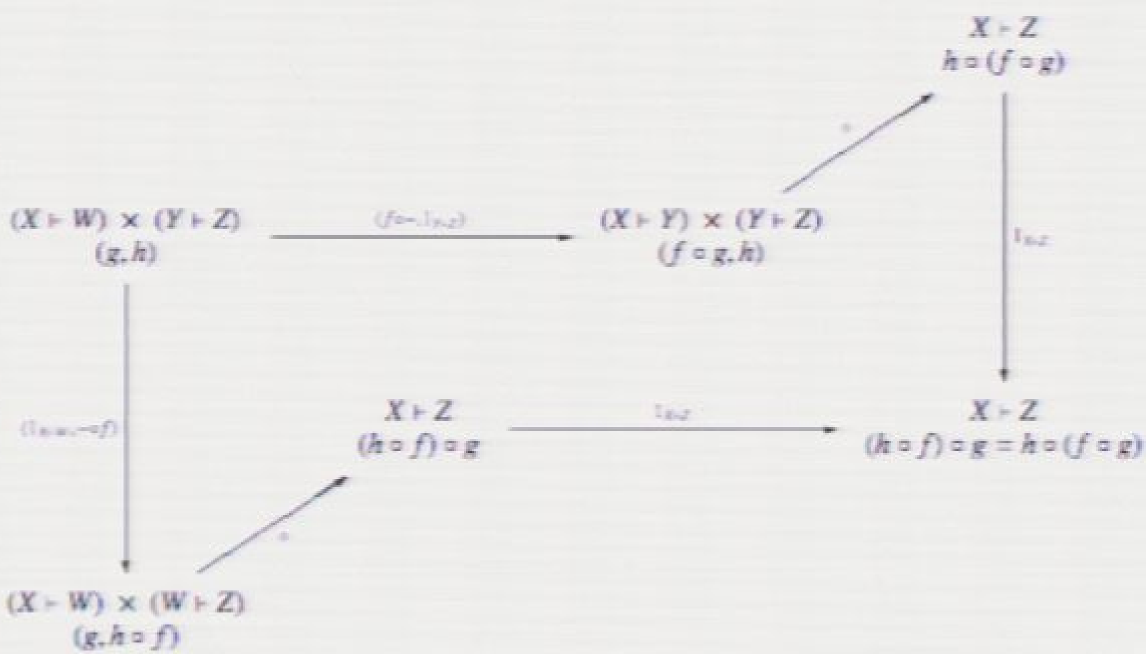


Figure 3: Dinaturality of the cut rule, where $f: W \to Y$, $g: X \to W$, $h: Y \to Z$.

4 Computation

4.1 Background

In the 1930s, while Turing was developing what are now called 'Turing machines' as a model for computation, Church and his student Kleene were developing a different model, called the 'lambda calculus' [29, 63]. While

Preview File Edit View Go Tools Bookmarks Window Help (3:11) Wed 12:42 PM

rosetta.pdf (page 50 of 73)

Previous Next Zoom Move Text Select Sidebar Search

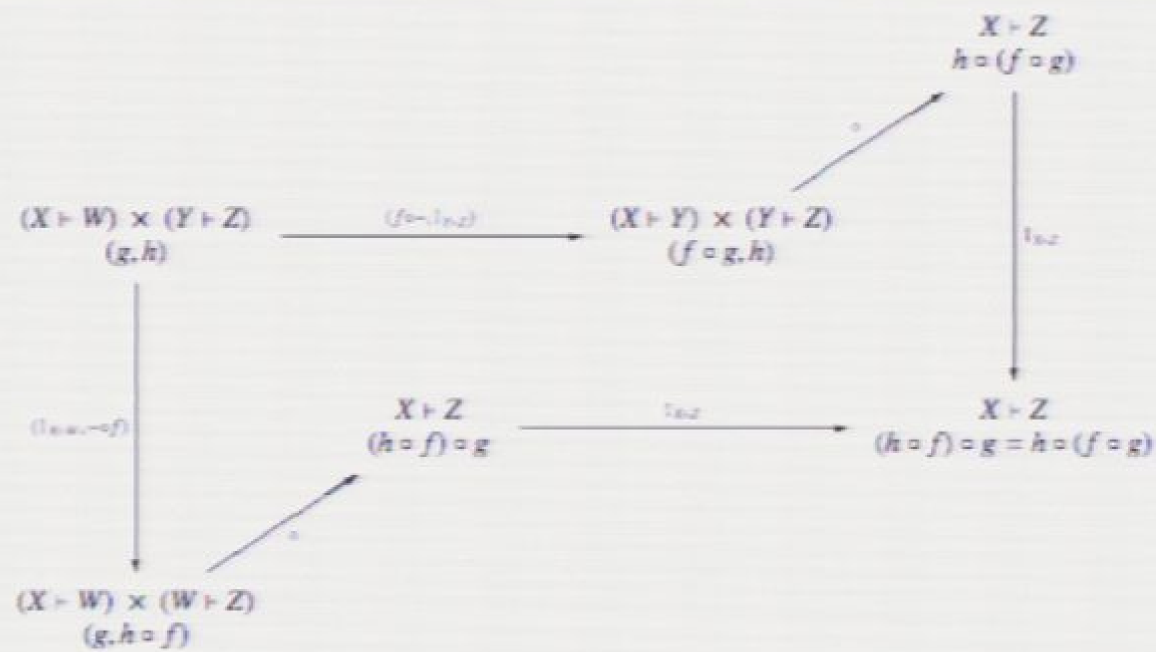


Figure 3: Dinaturality of the cut rule, where $f\colon W \to Y$, $g\colon X \to W$, $h\colon Y \to Z$.

4 Computation

4.1 Background

In the 1930s, while Turing was developing what are now called 'Turing machines' as a model for computation, Church and his student Kleene were developing a different model, called the 'lambda calculus' [29, 63]. While a Turing machine can be seen as an idealized, simplified model of computer *hardware*, the lambda calculus is more like a simple model of *software*.

By now the are many careful treatments of the lambda calculus in the literature, from Barendregt's magisterial tome [17] to the classic category-theoretic treatment of Lambek and Scott [67], to Hindley and Seldin's user-friendly introduction [51] and Selinger's elegant free online notes [86]. So, we shall content ourselves with a quick sketch.

Let us explain the meaning of application and lambda-abstraction. Application is simple. Since 'programs are data', we can think of any term either as a program or a piece of data. Since we can apply programs to data and get new data, we can apply any term f to any other term t and get a new term $f(t)$.

Lambda-abstraction is more interesting. We think of $(\lambda x.t)$ as the program that, given x as input, returns t as output. For example, consider

$$(\lambda x.x(x)).$$

This program takes any program x as input and returns $x(x)$ as output. In other words, it applies any program to itself. So, we have

$$(\lambda x.x(x))(s) = s(s)$$

for any term s.

More generally, if we apply $(\lambda x.t)$ to any term s, we should get back t, but with s substituted for each free occurrence of the variable x. This fact is codified in a rule called **beta reduction**:

$$(\lambda x.t)(s) = t[s/x]$$

where $t[s/x]$ is the term we get by taking t and substituting s for each free occurrence of x. But beware: this rule is not an equation in the usual mathematical sense. Instead, it is a 'rewrite rule': given the term on the left, we are allowed to rewrite it and get the term on the right. Starting with a term and repeatedly applying rewrite rules is how we take a program and let it run!

There are two other rewrite rules in the lambda calculus. If x is a variable and t is a term, the term

$$(\lambda x.t(x))$$

stands for the program that, given x as input, returns $t(x)$ as output. But this is just a fancy way of talking about the program t. So, the lambda calculus has a rewrite rule called **eta reduction**, saying

$$(\lambda x.t(x)) = t$$

input x it receives.

To get a feeling for how we can define arithmetic operations on Church numerals, consider

$$\lambda g.\bar{3}(\bar{2}(g)).$$

This program takes any program g, squares it, and then cubes the result. So, it raises g to the sixth power. This suggests that

$$\lambda g.\bar{3}(\bar{2}(g)) = \bar{6}.$$

Indeed this is true. If we treat the definitions of Church numerals as reversible rewrite rules, then we can start with the left side of the above equation and grind away using rewrite rules until we reach the right side:

$$\begin{array}{rcll}
(\lambda g.\bar{3}(\bar{2}(g))) & = & (\lambda g.\bar{3}((\lambda f.(\lambda x.f(f(x)))))(g)) & \text{def. of } \bar{2} \\
 & = & (\lambda g.\bar{3}(\lambda x.g(g(x)))) & \text{beta} \\
 & = & (\lambda g.(\lambda f.(\lambda x.f(f(f(x)))))(\lambda x.g(g(x)))) & \text{def. of } \bar{3} \\
 & = & (\lambda g.(\lambda x.(\lambda x.g(g(x)))((\lambda x.g(g(x)))((\lambda x.g(g(x)))(x))))) & \text{beta} \\
 & = & (\lambda g.(\lambda x.(\lambda x.g(g(x)))((\lambda g.g(g(x)))(g(g(x)))))) & \text{beta} \\
 & = & (\lambda g.(\lambda x.(\lambda x.g(g(x)))(g(g(g(g(x))))))) & \text{beta} \\
 & = & (\lambda g.(\lambda x.g(g(g(g(g(g(x)))))))) & \text{beta} \\
 & = & \bar{6} & \text{def. of } \bar{6}
\end{array}$$

If this calculation seems mind-numbing, that is precisely the point: it resembles the inner workings of a computer. We see here how the lambda calculus can serve as a programming language, with each step of computation corresponding to a rewrite rule.

Of course, we got the answer $\bar{6}$ because $3 \times 2 = 6$. Generalizing from this example, we can define a program called 'times' that multiplies Church numerals:

$$\text{times} = (\lambda a.(\lambda b.(\lambda x.a(b(x))))).$$

For example,

$$\text{times}(\bar{3})(\bar{2}) = \bar{6}.$$

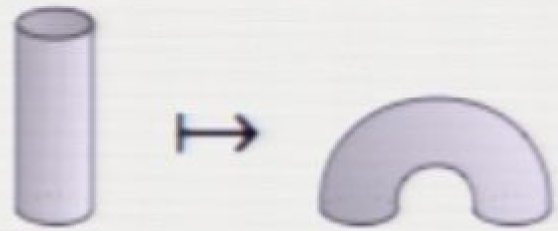
The enterprising reader can dream up similar programs for the other basic operations of arithmetic. With more cleverness, Church and Kleene were able to write terms corresponding to more complicated functions. They eventually came to believe that *all* computable functions $f\colon \mathbb{N} \to \mathbb{N}$ can be defined in the lambda calculus.

Meanwhile, Gödel was developing another approach to computability, the theory of 'recursive functions'. Around 1936, Kleene proved that the functions definable in the lambda calculus were the same as Gödel's

No Signal
VGA-1

No Signal

VGA-1

- **Topology:** 

- **Linear logic: IMPLIES** $\dfrac{X \otimes Y \vdash Z}{Y \vdash X \multimap Z}$ (c)

- **Computation: Currying**

```
z = f(x, y);
```

or

```
z = f(y)(x);
```

- **SET:** $f : X \times Y \to Z \quad \cong \quad f : Y \to Z^X$

Braided monoidal closed categories

- **Topology:** 
- **Linear logic: IMPLIES** $\dfrac{X \otimes Y \vdash Z}{Y \vdash X \multimap Z}$ (c)
- **Computation: Currying**

```
z = f(x, y);
```

or

```
z = f(y)(x);
```

- **SET:** $f : X \times Y \to Z \quad \cong \quad f : Y \to Z^X$

Model Theory (Quantum)

Syntax [Topology]	Semantics [QM]
manifold	Hilbert space of states
cobordism	linear transformation

Topological Quantum Field Theory