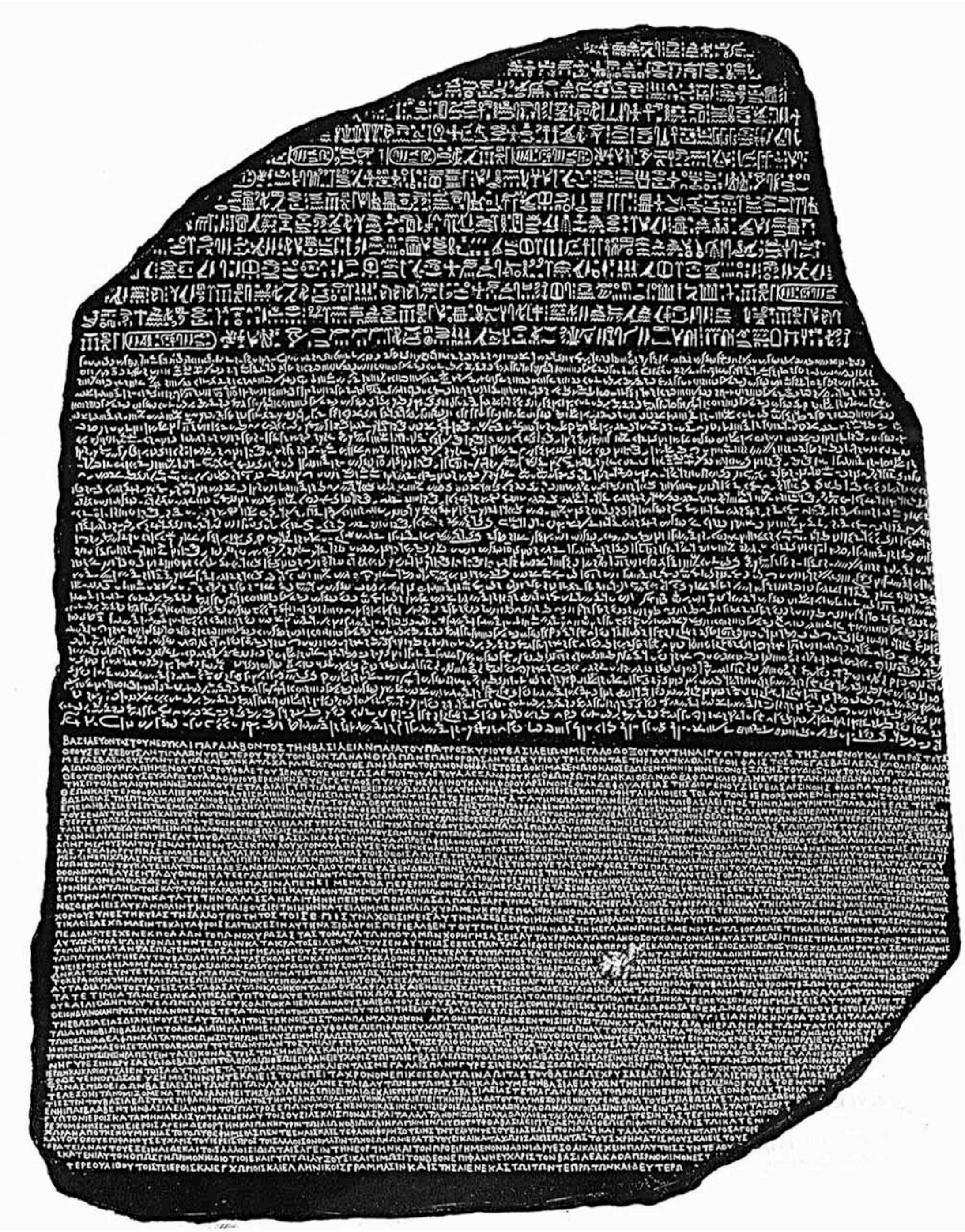
Physics, Topology, Logic, and Computation: A Rosetta Stone

John C. Baez
UC Riverside

Mike Stay
Google, U. of Auckland

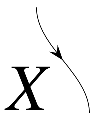


The Rosetta Stone (pocket version)

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 

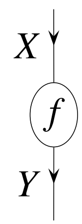
- **Quantum mechanics has Hilbert spaces:** $X \cong \mathbb{C}^n$
- **Topology has manifolds:** X 
- **Linear logic has propositions:**

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

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

Morphisms

- **String diagrams have vertices:**

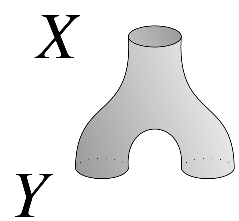


- **Quantum mechanics has linear transformations:**

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

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

Morphisms

- **Topology has cobordisms:** 

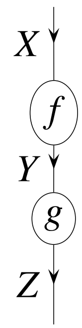
- **Linear logic has constructive proofs:**

$$X \vdash Y$$

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

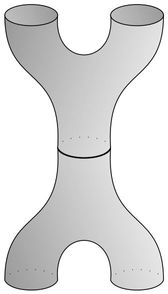
Morphisms compose associatively

- **String diagrams:**



- **Quantum mechanics: matrix multiplication**

- **Topology:**



Morphisms compose associatively

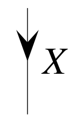
- **Linear logic:** $$\frac{Y \vdash Z \quad X \vdash Y}{X \vdash Z}\ (\circ)$$

- **Computation:**

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

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

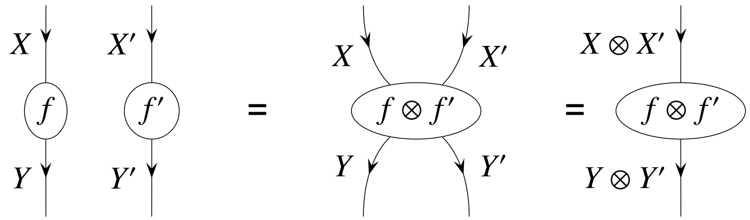
Identity morphisms

- **String diagrams:** 

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

Monoidal categories

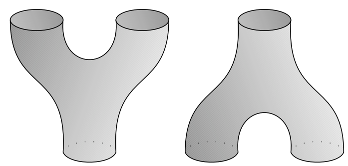
- **String diagrams:**



- **Quantum mechanics: tensor product**

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

Monoidal categories

- **Topology:** 

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

- **Computation: parallel programming**

```
Pair<X, X'> pair;
```

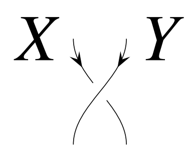
- **SET:** $f \times f' : X \times X' \to Y \times Y'$

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

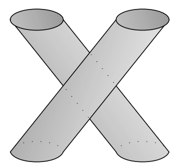
Braided monoidal categories

- **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.**

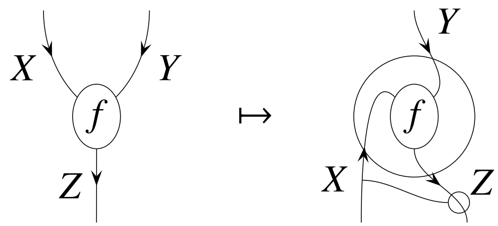
Braided monoidal categories

- **Topology:** 

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

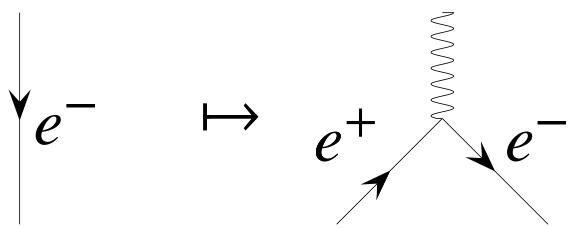
Braided monoidal closed categories

- **String diagrams:**

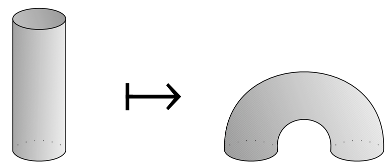


- **Quantum mechanics: antiparticles**

$$1_X : X \to X \quad \cong \quad \mathbf{pair} : I \to X^* \otimes X$$



Braided monoidal closed categories

- **Topology:** 

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

- **Computation: Currying**

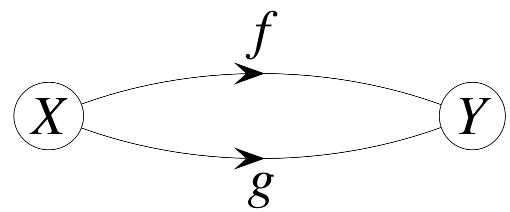
```
z = f(x, y);
```

or

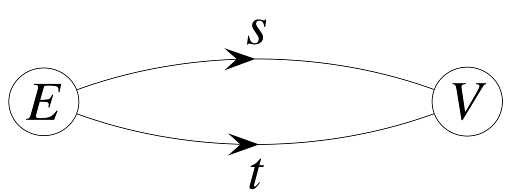
```
z = f(y)(x);
```

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

Model Theory



Model Theory



Th(Graph)

Model Theory

Th(Graph)	SET
object V	**set of vertices**
object E	**set of edges**
morphism $s : E \to V$	**function that picks out the source of each edge**
morphism $t : E \to V$	**function that picks out the target of each edge**

Model Theory (in Java)

```
interface ThGraph {
   // Internal interfaces
   interface V;
   interface E;

   // Methods
   V s(E);
   V t(E);
}
```

A functor is a structure-preserving map. In Java terms, a functor picks out a class that implements the interface.

Model Theory (Classical)

Syntax [Programming language]	Semantics [CE-SET]
data type	computably enumerable set of values
method	partially recursive function

Model Theory (Quantum)

Syntax [Programming language]	Semantics [QM]
data type	Hilbert space of values
method	linear transformation

Model Theory (Quantum)

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

Topological Quantum Field Theory