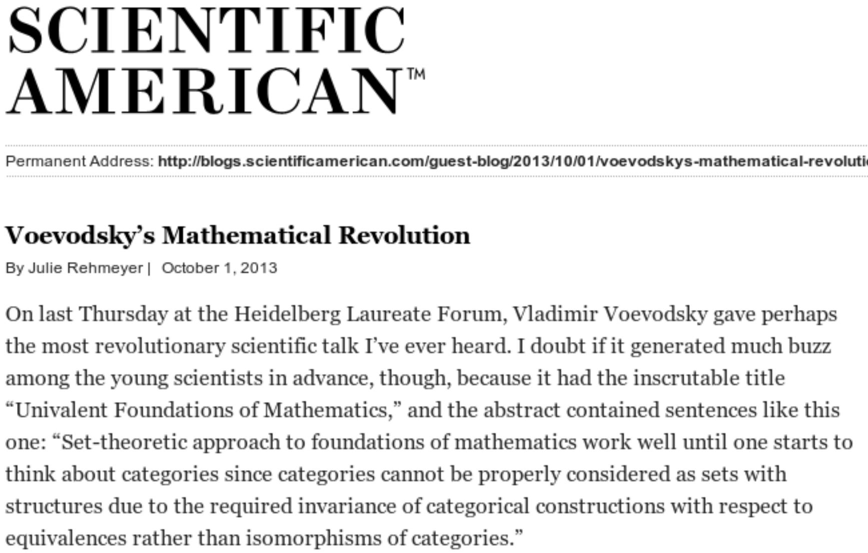
An Introduction to Homotopy Type Theory

Amr Sabry

School of Informatics and Computing
Indiana University

October 31, 2013



SCIENTIFIC AMERICAN™

Permanent Address: **http://blogs.scientificamerican.com/guest-blog/2013/10/01/voevodskys-mathematical-revoluti**

Voevodsky's Mathematical Revolution

By Julie Rehmeyer | October 1, 2013

On last Thursday at the Heidelberg Laureate Forum, Vladimir Voevodsky gave perhaps the most revolutionary scientific talk I've ever heard. I doubt if it generated much buzz among the young scientists in advance, though, because it had the inscrutable title "Univalent Foundations of Mathematics," and the abstract contained sentences like this one: "Set-theoretic approach to foundations of mathematics work well until one starts to think about categories since categories cannot be properly considered as sets with structures due to the required invariance of categorical constructions with respect to equivalences rather than isomorphisms of categories."

Homotopy Type Theory (HoTT)

- Foundational;
- New perspectives on old problems;
- Connections to computer science, logic, algebra, geometry, topology, and physics;
- Bridges between communities;
- What is it about exactly?

Steve Awodey's introduction (Feb. 2012)

Introduction

A new connection has recently come to light between Logic and Topology, namely an interpretation of the constructive type theory of Martin-Löf into homotopy theory.

1. Homotopy can be used as a tool to construct models of systems of logic.
2. Constructive type theory can be used as a formal calculus to reason about homotopy.
3. The computational implementation of type theory allows computer verified proofs in homotopy theory.
4. The homotopy interpretation suggests new logical constructions and axioms.
5. Voevodsky's *Univalent Foundations* program combines these aspects into a new program of foundations for mathematics.

In English (bigger informal picture)

- We are discovering that every process around us has computational content (even the design of outlines of 18th century string instruments as developed in Functional geometry and the Traité de Lutherie, ICFP 2013)
- We have known for quite a while (by the Curry-Howard correspondence) that mathematical proofs have computational content (type theory, mechanized logic, computational logic, etc. are all well-established)
- Feynman, Minsky, Landauer, Fredkin, and others have pushed the thesis that physical laws have computational content. Here is one of my favorite quotes:

> *. . . the real world is unlikely to supply us with unlimited memory or unlimited Turing machine tapes. Therefore, continuum mathematics is not executable, and physical laws which invoke that can not really be satisfactory . . .*

Common Themes: Physics perspective

- From the physical perspective: the laws of conservation of energy and information translate to “laws” requiring computation to preserve resources.
- Linear logic is a step in that direction but more generally one might argue that Computer Science is all about managing resources.
- The natural way of expressing computations that preserve resources is via type isomorphisms.

Common Themes: Mathematics perspective

- From the mathematical side: it is important to understand the computational content of propositional equality $\equiv$ as opposed to definitional equality $=$.
- If we claim that we have a proof that the proposition $x \equiv y$ is true, then it is interesting to be able to produce at least one such proof and to understand how various proofs might be connected.
- The natural way of expressing proofs of propositional equality is again via isomorphisms between propositions, which by the Curry-Howard correspondence is the same as type isomorphisms.

HoTT, informally

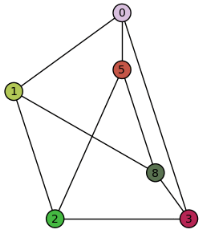
"Mathematics or Type Theory or Computation etc. with

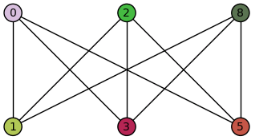
all equalities replaced by isomorphisms,

i.e., with equalities given computational content."

Isomorphisms

- When are two structures isomorphic?
- Example: are these two graphs isomorphic?

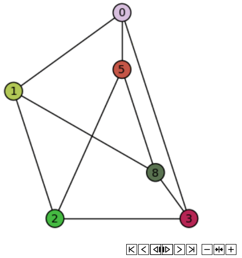


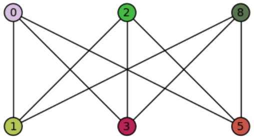


In general, you don't have nice colors and numbers to tell you which nodes correspond to which nodes.

Isomorphisms

Here is the proof that these two graphs are isomorphic.





[Credit to Pablo Angulo, using Sage software

`http://commons.wikimedia.org/wiki/File:Kuratowski.gif`]

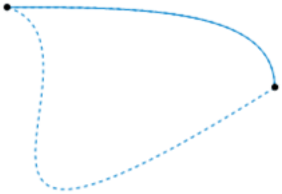
Isomorphisms

Two structures are isomorphic if we can morph one to the other by:

- bending,
- shrinking, and
- expanding.
- No cutting; no gluing
- Formally, a homotopy.

Homotopy

- Given two continuous functions from one topological space to another, a homotopy is a deformation of one function into the other.
- In homotopy type theory, we think of $x \equiv y$ as a path from point x to point y.
- Different proofs lead to different paths.
- A homotopy is a 2-path (a path between paths) that deforms one path to the other:



[Credit: Wikipedia `http://en.wikipedia.org/wiki/Homotopy`.]

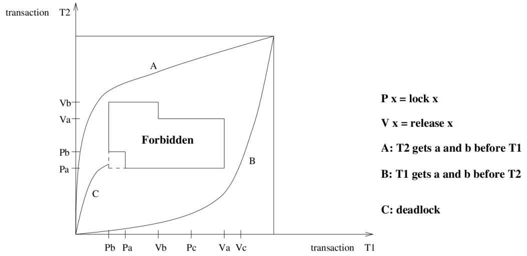
Mugs and donuts

- No talk that mentions topology can be complete without the obligatory coffee mug and donut.
- This is a homotopy of one space into another:

[Credit: Wikipedia `http://en.wikipedia.org/wiki/Homotopy`.]

Applications to Concurrency Theory

- Different computation schedules correspond to different paths.
- A homotopy between two paths means that the two corresponding schedules produce the same answers.
- Critical regions correspond to forbidden regions. A continuous deformation of paths cannot cross the forbidden region:



- The schedules on each side of the forbidden region might lead to different observable answers.

[Credit: Algebraic Topology and Concurrency, Fajstrup, Goubault, and Raussen, TCS 1998]

HoTT, a bit more formally

- (Martin-Löf) type theory
- Interpret types as topological spaces (or weak ∞-groupoids)
- Interpret identities as paths

Martin-Löf type theory

Universes $+$

Dependent functions $+$

Inductive data types

Martin-Löf type theory

Universes $+$

Dependent functions $+$

Inductive data types

in an executable framework. [We use Agda for this talk.]

Basic inductive types

```
- the empty type; the proposition FALSE
data ⊥ : Set where

- the type with one element
data ⊤ : Set where
  tt : ⊤

data Bool : Set where
  false   : Bool
  true    : Bool
```

Basic inductive types (ctd.)

```
data ℕ : Set where
  zero   : ℕ
  suc    : ℕ → ℕ

- disjoint unions or coproducts
data _⊎_ (A B : Set) : Set where
  inj₁ : A → A ⊎ B
  inj₂ : B → A ⊎ B


nb₁ : ℕ ⊎ Bool
nb₁ = inj₁ 3

nb₂ : ℕ ⊎ Bool
nb₂ = inj₂ true
```

Universes

- A universe is a “set” whose elements are “sets”.
- A universe is a “type” whose elements are “types”.
- There is no set of all sets, obviously.
- There is a hierarchy of universes

Universes (ctd.)

$n : \mathbb{N}$
$n = 5$

$t : \mathrm{Set}_0$ `- or just Set`
$t = \mathbb{N}$

$t' : \mathrm{Set}_1$
$t' = \mathrm{Set}_0$

$t'' : \mathrm{Set}_2$
$t'' = \mathrm{Set}_1$

Universes and families of sets

Families of sets or type families or dependent types:

```
P : ℕ → Set
P 0          = {!!}
P (suc n)    = {!!}

data Vec (A : Set) : ℕ → Set where
  []     : Vec A zero
  _::_   : ∀ {n} (x : A) (xs : Vec A n) → Vec A (suc n)

v₁ : Vec Bool 3
v₁ = false :: true :: false :: []
```

Dependent functions

```
_++_ : ∀ {m n} {A : Set} → Vec A m → Vec A n → Vec A (m + n)
[]          ++ ys = ys
(x :: xs)   ++ ys = x :: (xs ++ ys)

concat : ∀ {m n} {A : Set} → Vec (Vec A m) n → Vec A (n * m)
concat []            = []
concat (xs :: xss)   = xs ++ concat xss

c₁ : Vec Bool 6
c₁ = concat (   (false :: false :: [])   ::
                (true :: true :: [])     ::
                (false :: false :: [])   :: [] )
```

Dependent pairs

```
record Σ (A : Set) (B : A → Set) : Set where
  constructor _,_
  field
    proj₁ : A
    proj₂ : B proj₁


exT : Set
exT = Σ ℕ (λ n → Vec Bool n)

ex₁ : exT
ex₁ = (3 , (false :: true :: false :: []))

ex₂ : exT
ex₂ = (0 , [])
```

Pause

LaTeX crash . . .
Switch to second talk