Programming and Proving in Homotopy Type Theory

Daniel R. Licata

Carnegie Mellon University & Institute for Advanced Study
Sunkist Oranges

The Box

I Promised You from California.
Kepler Conjecture (1611)

No way to pack equally-sized spheres in space has higher density than
Hales’ proof (1998)

- Reduces Kepler Conjecture to proving that a function has a lower bound on 5,000 different configurations of spheres

- This requires solving 100,000 linear programming problems

- 1998 submission: 300 pages of math + 50,000 LOC (revised 2006: 15,000 LOC)
Proofs can be hard to check

In 2003, after 4 years’ work, 12 referees had checked lots of lemmas, but gave up on verifying the proof
Proofs can be hard to check

In 2003, after 4 years’ work, 12 referees had checked lots of lemmas, but gave up on verifying the proof.

“This paper has brought about a change in the journal’s policy on computer proof. It will no longer attempt to check the correctness of computer code.”
Computer-checked math

Hales’ proof of Kepler conjecture

Logic & Programming Language

Proof checker

Correct!

Incorrect
Computer-checked software

Your code, and proofs about it

Logic & Programming Language

Proof checker

Correct!
Incorrect
Computer-assisted proofs

**Proof assistant**

- Interactive proof editor
- Automated proofs
- Libraries
Computer-assisted proofs

- are much easier to believe:
  \textit{computer does the journal reviewing}

- can use computational methods
  and still be fully rigorous

- broaden access:
  \textit{computer as gifted\&talented teacher}

- are easier to write?
Kepler proof (85% done)

Informal

- 300 pages of math + 15,000 lines of code
- 15 hours to run

Computer-checked

- 350,000 lines of math + code
- >2 years to run
Kepler proof (85% done)

Informal

- 300 pages of math + 15,000 lines of code
- 15 hours to run

Computer-checked

- 350,000 lines of math + code
- >2 years to run

~5-10x longer
Kepler proof (85% done)

Informal
★ 300 pages of math + 15,000 lines of code
★ 15 hours to run

Computer-checked
★ 350,000 lines of math + code
★ >2 years to run
★ ~5-10x longer
★ ~2000x slower
Kepler proof (85% done)

**Informal**
- 300 pages of math + 15,000 lines of code
- 15 hours to run

**Computer-checked**
- 350,000 lines of math + code
- >2 years to run
  ~5-10x longer
  ~2000x slower

*We have some work to do!*
Now’s the time

Recent successes:

- Kepler conjecture [2013?, HOL Light]
- Four-color theorem [2005, Coq]
- Feit-Thompson theorem [2012, Coq]
- Correctness of a C compiler [2006, Coq]
- Correctness of Standard ML [2009, Twelf]

Mathematicians are interested!

- Year-long program at IAS hosted by Voevodsky
Making better proof assistants

**PL:** languages for expressing mathematics

**SE:** managing large codebases

**Compilers + distributed computing:** speed

**Machine learning:** automated proof search

**HCI:** usable by working mathematicians

**Graphics:** visualization
Making better proof assistants

**PL:** languages for expressing mathematics

**SE:** managing large codebases

Compilers + distributed computing: speed

Machine learning: automated proof search

**HCI:** usable by “working mathematicians”

Graphics: visualization
Homotopy Type Theory

Logic & Programming Language

Proof checker

Your proof

Correct!
Incorrect
Homotopy Type Theory

Your proof → Proof checker

Homotopy Type Theory

Correct! Incorrect
Type Theory

Basis of many successful proof assistants (Agda, Coq, NuPRL, Twelf)

- Functional programming language

  \[ \text{insertsort} : \text{list<int>} \rightarrow \text{list<int>} \]
  \[ \text{mergesort} : \text{list<int>} \rightarrow \text{list<int>} \]

- Unifies programming and proving:
  types are rich enough to do math/verification
Propositions as Types

1. A theorem is represented by a type
2. Proof is represented by a program of that type

\[ \forall x. \text{mergesort}(x) = \text{insertsort}(x) \]

*type of proofs of program equality*
Propositions as Types

1. A theorem is represented by a type
2. Proof is represented by a program of that type

proof : ∀x. mergesort(x) = insertsort(x)

*type of proofs of program equality*
1. A theorem is represented by a type
2. Proof is represented by a program of that type

proof : ∀x. mergesort(x) = insertsort(x)
proof x = case x of
    []  => reflexivity
    (x :: xs) => ...

proof by case analysis represented by a function defined by cases
Type are sets?

Traditional view:

**type theory**

<program> : <type>

<prog1> = <prog2>

**set theory**

x ∈ S

x = y
Type are sets?

Traditional view:

<table>
<thead>
<tr>
<th>type theory</th>
<th>set theory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;program&gt; : &lt;type&gt;</code></td>
<td><code>x ∈ S</code></td>
</tr>
<tr>
<td><code>&lt;prog1&gt; = &lt;prog2&gt;</code></td>
<td><code>x = y</code></td>
</tr>
</tbody>
</table>

In set theory, an equation is a proposition:

it holds or it doesn’t; we don’t ask why $1+1=2$
Type are sets?

Traditional view:

<table>
<thead>
<tr>
<th>type theory</th>
<th>set theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;program&gt; : &lt;type&gt;</td>
<td>x ∈ S</td>
</tr>
<tr>
<td>&lt;proof&gt; : &lt;prog1&gt; = &lt;prog2&gt;</td>
<td>x = y</td>
</tr>
</tbody>
</table>

In set theory, an equation is a proposition: it holds or it doesn’t; we don’t ask why $1+1=2$

In (intensional) type theory, an equation has a non-trivial <proof>
Homotopy Type Theory

category theory

type theory

homotopy theory
Types are $\infty$-groupoids

[Hofmann, Streicher, Awodey, Warren, Voevodsky, Lumsdaine, Gambino, Garner, van den Berg]
Types are $\infty$-groupoids

**Type theory**

<program> : <type>

<proof> : <prog₁> = <prog₂>

**Set theory**

$x \in S$

$x = y$
Types are $\infty$-groupoids

**type theory**

$<\text{program}> : <\text{type}>$

$<\text{proof}> : <\text{prog}_1> = <\text{prog}_2>$

$<2\text{-proof}> : <\text{proof}_1> = <\text{proof}_2>$

**set theory**

$x \in S$

$x = y$
Types are $\infty$-groupoids

**type theory**

\(<\text{program}> : \text{type}>\)

\(<\text{proof}> : <\text{prog}_1> = <\text{prog}_2>\)

\(<\text{2-proof}> : <\text{proof}_1> = <\text{proof}_2>\)

\(<\text{3-proof}> : <\text{2-proof}_1> = <\text{2-proof}_2>\)

**set theory**

\(x \in S\)

\(x = y\)
Types are $\infty$-groupoids

**type theory**

\begin{align*}
\langle \text{program} \rangle & : \langle \text{type} \rangle \\
\langle \text{proof} \rangle & : \langle \text{prog}_1 \rangle = \langle \text{prog}_2 \rangle \\
\langle 2\text{-proof} \rangle & : \langle \text{proof}_1 \rangle = \langle \text{proof}_2 \rangle \\
\langle 3\text{-proof} \rangle & : \langle 2\text{-proof}_1 \rangle = \langle 2\text{-proof}_2 \rangle \\
\vdots & \\
\vdots & \\
\end{align*}

**set theory**

\begin{align*}
x & \in S \\
x = y & \\
\end{align*}
Types are $\infty$-groupoids

**type theory**

\[
\langle \text{program} \rangle : \langle \text{type} \rangle \\
\langle \text{proof} \rangle : \langle \text{prog}_1 \rangle = \langle \text{prog}_2 \rangle \\
\langle 2\text{-proof} \rangle : \langle \text{proof}_1 \rangle = \langle \text{proof}_2 \rangle \\
\langle 3\text{-proof} \rangle : \langle 2\text{-proof}_1 \rangle = \langle 2\text{-proof}_2 \rangle \\
\vdots
\]

**set theory**

\[
x \in S \\
x = y
\]

Proofs, 2-proofs, 3-proofs, … all influence how a program runs
Types are $\infty$-groupoids

**type theory**

\[
\begin{align*}
\langle \text{program} \rangle & : \langle \text{type} \rangle \\
\langle \text{proof} \rangle & : \langle \text{prog}_1 \rangle = \langle \text{prog}_2 \rangle \\
\langle 2\text{-proof} \rangle & : \langle \text{proof}_1 \rangle = \langle \text{proof}_2 \rangle \\
\langle 3\text{-proof} \rangle & : \langle 2\text{-proof}_1 \rangle = \langle 2\text{-proof}_2 \rangle \\
& \vdots
\end{align*}
\]

**set theory**

\[
\begin{align*}
x & \in S \\
x & = y
\end{align*}
\]

$\infty$-groupoid: each level has a group structure, and they interact

Proofs, 2-proofs, 3-proofs, … all influence how a program runs
Homotopy Type Theory

- new programs
- category theory
- type theory
- homotopy theory
- new possibilities for computer-checked proofs
I am developing a computational theory of $\infty$-groupoids and applying it to computer-checked math and software.
Results

1. I have developed computer-checked proofs of theorems in homotopy theory [LICS’13]

2. I have discovered how to run programs in Homotopy Type Theory, for the special case of 2-dimensional type theory [POPL’12]

3. I have applied these new concepts to computer-checked software [thesis + MFPS’11]
Outline

1. Computer-checked homotopy theory
2. Computer-checked software
Outline

1. Computer-checked homotopy theory
2. Computer-checked software
Homotopy Theory

A branch of topology, the study of spaces and continuous deformations

[image from wikipedia]
Homotopy Theory

A branch of topology, the study of spaces and continuous deformations

[Image from wikipedia]
Synthetic vs Analytic

Synthetic geometry (Euclid)

POSTULATES.

I. Let it be granted that a straight line may be drawn from any one point to any other point.

II. That a terminated straight line may be produced to any length in a straight line.

III. And that a circle may be described from any centre, at any distance from that centre.

Analytic geometry (Descartes)

[Image from wikipedia]
Synthetic vs Analytic

**Synthetic geometry (Euclid)**

**Analytic geometry (Descartes)**

Classical homotopy theory is analytic:

- a space is a set of points equipped with a topology
- a path is a set of points, given continuously

[Image from wikipedia]
Synthetic homotopy theory

**homotopy theory**
- space
- points
- paths
- homotopies

**type theory**
- \(<\text{type}\>
- \(<\text{program}\> : <\text{type}\>
- \(<\text{proof}\> : <\text{prog}_1> = <\text{prog}_2>
- \(<\text{2-proof}\> : <\text{proof}_1> = <\text{proof}_2>\)
Synthetic homotopy theory

<table>
<thead>
<tr>
<th>homotopy theory</th>
<th>type theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>&lt;type&gt;</td>
</tr>
<tr>
<td>points</td>
<td>&lt;program&gt; : &lt;type&gt;</td>
</tr>
<tr>
<td>paths</td>
<td>&lt;proof&gt; : &lt;prog1&gt; = &lt;prog2&gt;</td>
</tr>
<tr>
<td>homotopies</td>
<td>&lt;2-proof&gt; : &lt;proof1&gt; = &lt;proof2&gt;</td>
</tr>
</tbody>
</table>

A path is not a set of points; it is a primitive notion.
Spaces as types
Spaces as types

A space is a type $A$
Spaces as types

A space is a type $A$

points are programs $M : A$
Spaces as types

A space is a type $A$

Points are programs $M : A$

Paths are proofs of equality $\alpha : M =_A N$
Spaces as types

A space is a type \( A \)

Points are programs \( M : A \)

Paths are proofs of equality \( \alpha : M =_A N \)

Path operations
Spaces as types

a space is a type $A$

points are programs $M : A$

paths are proofs of equality $\alpha : M =_A N$

path operations

$id : M = M \text{ (refl)}$
Spaces as types

a space is a type $A$

points are programs $M : A$

paths are proofs of equality $\alpha : M =_A N$

path operations

$id : M = M$ (refl)

$\alpha^{-1} : N = M$ (sym)
Spaces as types

A space is a type $A$

Paths are proofs of equality

$\alpha : M =_A N$

Path operations

$id : M = M$ (refl)
$\alpha^{-1} : N = M$ (sym)
$\beta \circ \alpha : M = P$ (trans)

Points are programs

$M : A$
Spaces as types

A space is a type $A$

Points are programs $M : A$

Paths are proofs of equality $\alpha : M =_A N$

Path operations
- $\text{id} : M = M$ (refl)
- $\alpha^{-1} : N = M$ (sym)
- $\beta \circ \alpha : M = P$ (trans)

Fundamental group: group of loops
Spaces as types

a space is a type $A$

points are programs
$M : A$

paths are proofs of equality
$\alpha : M =_A N$

path operations

$\text{id} : M = M$ (refl)
$\alpha^{-1} : N = M$ (sym)
$\beta \circ \alpha : M = P$ (trans)

Fundamental group:

group of loops
modulo homotopy
Homotopy

Deformation of one path into another

$\alpha$

$\beta$
Homotopy

Deformation of one path into another
Homotopy

Deformation of one path into another

[Image from wikipedia]
Homotopy

Deformation of one path into another

= 2-dimensional *path between paths*
Homotopy

Deformation of one path into another

\[ \alpha \approx \beta \]

= 2-dimensional path between paths
Homotopy

Deformation of one path into another

= 2-dimensional path between paths

Homotopy theory is the study of spaces by way of their paths, homotopies, homotopies between homotopies, ....
We can do homotopy theory by writing functional programs
Functions on sets

Function on a set gives the image of each element:

\[ \text{not} : \text{Bool} \rightarrow \text{Bool} \]
\[ \text{not(true)} = \text{false} \]
\[ \text{not(false)} = \text{true} \]
Functions on spaces

Function on a space gives the image of each point

Circle → Circle

loop

base

base
Functions on spaces

Function on a space gives the image of each point and each path!
Functions on spaces

Function on a space gives the image of each point and each path!
Functions on spaces

Function on a space gives the image of each point and each path!
Circle Recursion

reverse : Circle \rightarrow Circle
reverse(base) = base
reverse(loop) = loop^{-1}
Circle Recursion

reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop$^{-1}$

This specifies the image for all paths because

1. circle is inductively generated by loop: all paths are built from loop by identity, inverse, composition
2. all functions are homomorphisms
Homomorphism

\[ \text{reverse} : \text{Circle} \rightarrow \text{Circle} \]
\[ \text{reverse}(\text{base}) = \text{base} \]
\[ \text{reverse}(\text{loop}) = \text{loop}^{-1} \]

Computation steps:
\[ \text{reverse}(\text{loop} \circ \text{loop}) \]
Homomorphism

reverse : Circle $\to$ Circle
reverse(base) = base
reverse(loop) = loop$^{-1}$

Computation steps:
reverse(loop o loop) = (reverse loop) o (reverse loop) homomorphism
Homomorphism

\[
\text{reverse : Circle} \to \text{Circle}
\]

\[
\text{reverse}(\text{base}) = \text{base}
\]

\[
\text{reverse}(\text{loop}) = \text{loop}^{-1}
\]

**Computation steps:**

\[
\text{reverse}(\text{loop} \circ \text{loop})
\]

\[
= (\text{reverse loop}) \circ (\text{reverse loop})
\]

\[
= \text{loop}^{-1} \circ \text{loop}^{-1}
\]
Homomorphism

reverse : Circle \rightarrow Circle
reverse(base) = base
reverse(loop) = loop^{-1}

Computation steps:
reverse(loop o loop)
= (reverse loop) o (reverse loop)
= loop^{-1} o loop^{-1}
= (loop o loop)^{-1}
Homomorphism

\[\text{reverse} : \text{Circle} \to \text{Circle}\]
\[\text{reverse}(\text{base}) = \text{base}\]
\[\text{reverse}(\text{loop}) = \text{loop}^{-1}\]

**Computation steps:**
\[\text{reverse}(\text{loop} \circ \text{loop})\]
\[= (\text{reverse loop}) \circ (\text{reverse loop})\]
\[= \text{loop}^{-1} \circ \text{loop}^{-1}\]
\[= (\text{loop} \circ \text{loop})^{-1}\]
Circle induction

reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop^{-1}

Theorem: ∀p. reverse(p) = p^{-1}

Proof: uses circle induction:
To prove a predicate P for all points on the circle,
suffices to prove P(base),
continuously in the loop
We can do interesting homotopy theory synthetically.
Telling spaces apart
Telling spaces apart

fundamental group is non-trivial \((\mathbb{Z} \times \mathbb{Z})\)

\(\nless\)

fundamental group is trivial
Homotopy Groups

*Homotopy groups of a space $X$:

* $\pi_1(X)$ is fundamental group (group of loops)
* $\pi_2(X)$ is group of *homotopies* (2-dimensional loops)
* $\pi_3(X)$ is group of 3-dimensional loops
* ...
Homotopy groups

$k^{th}$ homotopy group

[n-dimensional sphere]

[Image from wikipedia]
Computer-checked proofs

$k^{\text{th}}$ homotopy group

n-dimensional sphere

[Image from Wikipedia]
Computer-checked proofs

$k^{\text{th}}$ homotopy group

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$n_4$</th>
<th>$n_5$</th>
<th>$n_6$</th>
<th>$n_7$</th>
<th>$n_8$</th>
<th>$n_9$</th>
<th>$n_{10}$</th>
<th>$n_{11}$</th>
<th>$n_{12}$</th>
<th>$n_{13}$</th>
<th>$n_{14}$</th>
<th>$n_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^1$</td>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^2$</td>
<td>0</td>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^3$</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^5$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^7$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S^8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>$Z_2$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

[image from wikipedia]
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)

2. $\pi_k(S^n)$ trivial for $k < n$

3. Freudenthal suspension theorem
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)

4. Eilenberg-Mac Lane spaces $K(G,n)$
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem  
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$

∗ 11,000 lines of Agda code (most since January)
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$

* 11,000 lines of Agda code (most since January)
* Proofs are programs: you can run them
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$

- 11,000 lines of Agda code (most since January)
- Proofs are programs: you can run them
- Computer-checked proofs shorter than “informalized”
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem  
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$

- 11,000 lines of Agda code (most since January)
- Proofs are programs: you can run them
- Computer-checked proofs shorter than “informalized”
- Proofs are new: I discovered a type-theoretic method that is used in all of these proofs
Computer-checked proofs

1. $\pi_n(S^n) = \mathbb{Z}$ (w/ G. Brunerie)
2. $\pi_k(S^n)$ trivial for $k < n$
3. Freudenthal suspension theorem
   (w/ P. Lumsdaine; Blakers-Massey w.i.p)
4. Eilenberg-Mac Lane spaces $K(G,n)$

![Diagram showing the homotopy groups of spheres]

[LIACS’13]
Fundamental group of circle

Two functions:
1. $\text{winding} : (\text{base} = \text{base}) \rightarrow \mathbb{Z}$
2. $\text{loop}^n : \mathbb{Z} \rightarrow (\text{base} = \text{base})$

Three proofs:
1. $\forall n : \mathbb{Z}. \text{winding}(\text{loop}^n) = n$
2. $\forall p. \text{loop}^{\text{winding}(p)} = p$
3. $\forall n,m. \text{loop}^{n+m} = \text{loop}^n \circ \text{loop}^m$
Fundamental group of circle

Two functions:

1. \( \text{winding} : (\text{base} = \text{base}) \rightarrow \mathbb{Z} \)
2. \( \text{loop}^n : \mathbb{Z} \rightarrow (\text{base} = \text{base}) \)

Three proofs:

1. \( \forall n : \mathbb{Z} . \:\text{winding}(\text{loop}^n) = n \)
2. \( \forall p . \:\text{loop}^{\text{winding}(p)} = p \)
3. \( \forall n, m . \:\text{loop}^{n+m} = \text{loop}^n \circ \text{loop}^m \)
Fundamental group of circle

Two functions:
1. \( \text{winding} : (\text{base} = \text{base}) \rightarrow \mathbb{Z} \) (uses circle recursion)
2. \( \text{loop}^n : \mathbb{Z} \rightarrow (\text{base} = \text{base}) \)

Three proofs:
1. \( \forall n : \mathbb{Z}. \text{winding}(\text{loop}^n) = n \)
2. \( \forall p. \text{loop}^{\text{winding}(p)} = p \)
3. \( \forall n, m. \text{loop}^{n+m} = \text{loop}^n \circ \text{loop}^m \) (induction principles for circle, paths, int; and calculations using my computational interpretation)
Fundamental group of the circle

Informal

Computer-checked
Outline

1. Computer-checked homotopy theory
2. Computer-checked software
Example

Convert dates between European and US formats, inside a data structure

`[{key=4,n=“John”}, d=(30,5,1956)],
{key=8,n=“Hugo”}, d=(29,12,1978)],
{key=15,n=“James”}, d=(1,7,1968)],
{key=16,n=“Sayid”}, d=(2,10,1967)],
{key=23,n=“Jack”}, d=(3,12,1969)],
{key=42,n=“Sun”}, d=(20,3,1980)]`

`[{key=4,n=“John”}, d=(5,30,1956)],
{key=8,n=“Hugo”}, d=(12,29,1978)],
{key=15,n=“James”}, d=(7,1,1968)],
{key=16,n=“Sayid”}, d=(10,2,1967)],
{key=23,n=“Jack”}, d=(12,3,1969)],
{key=42,n=“Sun”}, d=(3,20,1980)]`

**Spec:** Conversion is a bijection: converting back and forth doesn’t change the data.
Type theory

\[
\text{conv1 : (Nat \times \text{String} \times ((\text{Nat} \times \text{Nat}) \times \text{Nat}))} \\
\quad \quad \quad \rightarrow (\text{Nat} \times \text{String} \times ((\text{Nat} \times \text{Nat}) \times \text{Nat})) \\
\text{conv1 (key, name, ((x, y), year)) = (key, name, ((y, x), year))}
\]

\[
\text{convert : DB \rightarrow DB} \\
\text{convert = map conv1}
\]

\[
\begin{align*}
\text{map-fusion : \forall \{A B C\} (g : B \rightarrow C) } \\
\quad \quad \quad \quad (f : A \rightarrow B) (l : \text{List} A) \\
\quad \quad \quad \quad \quad \rightarrow \text{map} (g \circ f) l = \text{map} g (\text{map} f l) \\
\text{map-fusion} g f \square = \text{id} \\
\text{map-fusion} g f (x :: xs) = \\
\quad \quad \text{ap} (_ :: _) (g (f x)) (\text{map-fusion} f g xs)
\end{align*}
\]

\[
\begin{align*}
\text{map-idfunc : \forall \{A\} (l : \text{List} A) \rightarrow \text{map} (\lambda x \rightarrow x) l = l \\
\text{map-idfunc} \square = \text{id} \\
\text{map-idfunc} (x :: xs) = \text{ap} (_ :: x) (\text{map-idfunc} xs)
\end{align*}
\]

\[
\begin{align*}
\text{convert-inv : convert \circ convert = (\lambda x \rightarrow x)} \\
\text{convert-inv = map conv1 \circ map conv1} \\
\quad \quad = (\lambda (\text{map-fusion} \text{conv1 conv1})) \\
\quad \quad = (\lambda (\text{id})) \\
\quad \quad = (\lambda (\text{map-idfunc})) \\
\quad \quad (\lambda x \rightarrow x)
\end{align*}
\]

\[
\begin{align*}
\text{convert-bijection : Bijective DB DB} \\
\text{convert-bijection =} \\
\quad (\text{convert, is-bijection convert} \\
\quad (\lambda x \rightarrow (\text{ap convert-inv})) \\
\quad (\lambda x \rightarrow (\text{ap convert-inv})))
\end{align*}
\]

Homotopy Type Theory

\[
\text{swapf : (Nat \times Nat) \rightarrow (Nat \times Nat)} \\
\text{swapf (x, y) = (y, x)}
\]

\[
\begin{align*}
\text{swap : Bijective (Nat \times Nat) (Nat \times Nat)} \\
\text{swap = (swapf, } \\
\quad \text{is-bijection swapf (\lambda _ \rightarrow \text{id}) (\lambda _ \rightarrow \text{id}) })
\end{align*}
\]

\[
\text{There : Type \rightarrow Type} \\
\text{There A = List (Nat \times \text{String} \times A \times A )}
\]

\[
\begin{align*}
\text{convert : DB \rightarrow DB} \\
\text{convert = cast There swap}
\end{align*}
\]

\[
\begin{align*}
\text{convert-bijection : Bijective DB DB} \\
\text{convert-bijection =} \\
\quad (\text{convert, cast-is-bijection There swap})
\end{align*}
\]
convert

```json
[{key=4,n="John", d=(30,5,1956)},
 {key=8,n="Hugo",d=(29,12,1978)},
 {key=15,n="James”,d=(1,7,1968)},
 {key=16,n="Sayid”,d=(2,10,1967)},
 {key=23,n="Jack”,d=(3,12,1969)},
 {key=42,n="Sun”,d=(20,3,1980)]
```

```json
[{key=4,n="John”,d=(5,30,1956)},
 {key=8,n="Hugo”,d=(12,29,1978)},
 {key=15,n="James”,d=(7,1,1968)},
 {key=16,n="Sayid”,d=(10,2,1967)},
 {key=23,n="Jack”,d=(12,3,1969)},
 {key=42,n="Sun”,d=(3,20,1980)]
```
1. Define a function

\[ \text{swap}(x, y) = (y, x) \]
1. Define a function
   \[ \text{swap}(x, y) = (y, x) \]

2. Prove that \text{swap} is a bijection (it’s self-inverse)
1. Define a function
   \[\text{swap}(x, y) = (y, x)\]

2. Prove that \text{swap} is a bijection (it’s self-inverse)

3. Define a \textit{parametrized type} describing where to swap:
   \[\text{There}(X) = \text{List}\{\text{key}: \text{int}, \text{n}: \text{string}, \text{d}: X \times \text{int}\}\]
1. Define a function
   \( \text{swap}(x, y) = (y, x) \)

2. Prove that \text{swap} is a bijection (it’s self-inverse)

3. Define a \textit{parametrized type} describing where to swap:
   \[
   \text{There}(X) = \text{List}\{ \text{key: int, n: string, d: } X \times \text{int} \}\]

4. Define
   \[
   \text{convert}(\text{db}) = \text{cast}_{\text{There}}(\text{swap}, \text{db})
   \]
Types write code and proofs for you
Functions on spaces

Function on a space gives the image of each point

and each path!
Functions on types
There(\(X\))=List\{key:int, n:string, d: X\times int\} is a function on the space of types, so it must also give an image for each path between types.
1. \( \text{There}(X) = \text{List}\{\text{key: int, n: string, d: X x int}\} \) is a function on the **space of types**, so it must also give an image for each path between types.

2. We define the paths between types to be **bijections**.
1. There(\(X\)) = \{\text{key: int, n: string, d: } X \times \text{int}\} is a function on the space of types, so it must also give an image for each path between types.

2. We define the paths between types to be bijections.

3. \(\therefore\) There gives an image for swap.
**Cast**

\[ \text{cast}_{\text{There}}(\text{swap}) \]

applies \text{There} to swap: a \textit{type-directed program} that builds bigger bijections from smaller ones
Computational interpretation of cast:

There(X)=List\{key:int, n:string, d:X\times int\}

castThere(swap,db)

[[key=4,n="John", d=(30,5,1956)],
 {key=8,n="Hugo",d=(29,12,1978)},
 {key=15,n="James",d=(1,7,1968)},
 {key=16,n="Sayid",d=(2,10,1967)},
 {key=23,n="Jack",d=(3,12,1969)},
 {key=42,n="Sun",d=(20,3,1980)]]

[[key=4,n="John",d=(5,30,1956)],
 {key=8,n="Hugo",d=(12,29,1978)},
 {key=15,n="James",d=(7,1,1968)},
 {key=16,n="Sayid",d=(10,2,1967)},
 {key=23,n="Jack",d=(12,3,1969)},
 {key=42,n="Sun",d=(3,20,1980)]]
Computational interpretation of cast:

\[ \text{There}(X) = \text{List}\{\text{key:int, n:string, d:X\times\text{int}}\} \]

\[
\text{cast} \text{There}(\text{swap,db}) = \text{map (cast} \text{There}_1 \text{ swap) db} \]

\[
[[\text{key}=4, \text{n}=\text{“John”}, \text{d}=(30,5,1956)], \\
[\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(29,12,1978)], \\
[\text{key}=15, \text{n}=\text{“James”}, \text{d}=(1,7,1968)], \\
[\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(2,10,1967)], \\
[\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(3,12,1969)], \\
[\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(20,3,1980)]]
\]

\[
[[\text{key}=4, \text{n}=\text{“John”}, \text{d}=(5,30,1956)], \\
[\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(12,29,1978)], \\
[\text{key}=15, \text{n}=\text{“James”}, \text{d}=(7,1,1968)], \\
[\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(10,2,1967)], \\
[\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(12,3,1969)], \\
[\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(3,20,1980)]]
\]
Computational interpretation of cast:

\[ \text{cast}_{\text{There1}}(\text{swap, db}) = \text{map} \left( \text{cast}_{\text{There1}} \text{swap} \right) \text{db} \]

\[
\begin{align*}
\{\text{key}=4, \text{n}=\text{“John”}, \text{d}=(30, 5, 1956)\}, \\
\{\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(29, 12, 1978)\}, \\
\{\text{key}=15, \text{n}=\text{“James”}, \text{d}=(1, 7, 1968)\}, \\
\{\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(2, 10, 1967)\}, \\
\{\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(3, 12, 1969)\}, \\
\{\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(20, 3, 1980)\}\end{align*}
\]

\[
\begin{align*}
\{\text{key}=4, \text{n}=\text{“John”}, \text{d}=(5, 30, 1956)\}, \\
\{\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(12, 29, 1978)\}, \\
\{\text{key}=15, \text{n}=\text{“James”}, \text{d}=(7, 1, 1968)\}, \\
\{\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(10, 2, 1967)\}, \\
\{\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(12, 3, 1969)\}, \\
\{\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(3, 20, 1980)\}\end{align*}
\]
Computational interpretation of cast:

There1(X) = {key:int, n:string, d:X×int}

cast₁₈There(swap, db)
= map (cast₁₈There swap) db
= map ({key,n,(d,m,y)} =>
  {key,n,( ,y)}) db

[[key=4,n="John", d=(30,5,1956)],
 {key=8,n="Hugo", d=(29,12,1978)],
 {key=15,n="James", d=(1,7,1968)],
 {key=16,n="Sayid", d=(2,10,1967)],
 {key=23,n="Jack", d=(3,12,1969)],
 {key=42,n="Sun", d=(20,3,1980)]}
Computational interpretation of cast:

There1(X)={key:int, n:string, d:Xxint}

\[
\text{cast}_{\text{There1}}(\text{swap,db}) = \text{map (cast}_{\text{There1}} \text{ swap) db}
\]

\[
= \text{map } (\{\text{key,n,}(d,m,y)\} =>
\{\text{key,n,}(\text{cast}_{\text{Here}}(\text{swap,}(d,m)),y)\}) \text{ db}
\]

\{
\{key=4,n=“John”, d=(30,5,1956)},
\{key=8,n=“Hugo”, d=(29,12,1978)},
\{key=15,n=“James”, d=(1,7,1968)},
\{key=16,n=“Sayid”, d=(2,10,1967)},
\{key=23,n=“Jack”, d=(3,12,1969)},
\{key=42,n=“Sun”, d=(20,3,1980)}\}

\{
\{key=4,n=“John”, d=(5,30,1956)},
\{key=8,n=“Hugo”, d=(12,29,1978)},
\{key=15,n=“James”, d=(7,1,1968)},
\{key=16,n=“Sayid”, d=(10,2,1967)},
\{key=23,n=“Jack”, d=(12,3,1969)},
\{key=42,n=“Sun”, d=(3,20,1980)}\}
Computational interpretation of cast:

\[ \text{Here}(X) = X \]

\[
\text{cast}_{\text{There}}(\text{swap}, \text{db}) \\
= \text{map} \left( \text{cast}_{\text{There1}} \text{ swap} \right) \text{ db} \\
= \text{map} \left( \{\text{key}, \text{n}, (\text{d}, \text{m}, \text{y})\} \rightarrow \{\text{key}, \text{n}, (\text{cast}_{\text{Here}} \text{ (swap}, (\text{d}, \text{m})), \text{y})\} \right) \text{ db}
\]

\[
[\{\text{key}=4, \text{n}=\text{“John”}, \text{d}=(30,5,1956)}, \{\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(29,12,1978)}, \{\text{key}=15, \text{n}=\text{“James”}, \text{d}=(1,7,1968)}, \{\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(2,10,1967)}, \{\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(3,12,1969)}, \{\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(20,3,1980)}]\]

\[
[\{\text{key}=4, \text{n}=\text{“John”}, \text{d}=(5,30,1956)}, \{\text{key}=8, \text{n}=\text{“Hugo”}, \text{d}=(12,29,1978)}, \{\text{key}=15, \text{n}=\text{“James”}, \text{d}=(7,1,1968)}, \{\text{key}=16, \text{n}=\text{“Sayid”}, \text{d}=(10,2,1967)}, \{\text{key}=23, \text{n}=\text{“Jack”}, \text{d}=(12,3,1969)}, \{\text{key}=42, \text{n}=\text{“Sun”}, \text{d}=(3,20,1980)}]\]
Computational interpretation of cast:

**Here**\(X) = X

\[
\text{cast}_\text{There}(\text{swap,db}) = \text{map} \ (\text{cast}_\text{There1} \ \text{swap}) \ \text{db}
\]

\[
= \text{map} \ (\{\text{key},n,(d,m,y)\} \Rightarrow \{\text{key},n, (\text{cast}_\text{Here}(\text{swap},(d,m)),y)\}) \ \text{db}
\]

\[
= \text{map} \ (\{\text{key},n,(d,m,y)\} \Rightarrow \{\text{key},n,(m,d,y)\}) \ \text{db}
\]

\[
[\{\text{key}=4, n=\text{“John”}, d=(30,5,1956)\}, \{\text{key}=8, n=\text{“Hugo”}, d=(29,12,1978)\}, \{\text{key}=15, n=\text{“James”}, d=(1,7,1968)\}, \{\text{key}=16, n=\text{“Sayid”}, d=(2,10,1967)\}, \{\text{key}=23, n=\text{“Jack”}, d=(3,12,1969)\}, \{\text{key}=42, n=\text{“Sun”}, d=(20,3,1980)\}]
\]

\[
[\{\text{key}=4, n=\text{“John”}, d=(5,30,1956)\}, \{\text{key}=8, n=\text{“Hugo”}, d=(12,29,1978)\}, \{\text{key}=15, n=\text{“James”}, d=(7,1,1968)\}, \{\text{key}=16, n=\text{“Sayid”}, d=(10,2,1967)\}, \{\text{key}=23, n=\text{“Jack”}, d=(12,3,1969)\}, \{\text{key}=42, n=\text{“Sun”}, d=(3,20,1980)\}]
\]
Type theory

conv1 : (Nat x String x ((Nat x Nat) x Nat)) → (Nat x String x ((Nat x Nat) x Nat))
conv1 (key, name, ((x, y), year)) = (key, name, ((y, x), year))

convert : DB → DB
convert = map conv1

map-fusion : ∀ {A B C} (g : B → C)
(f : A → B) (l : List A)
→ map (g o f) l = map g (map f l)
map-fusion g f [] = id
map-fusion g f (x :: xs) = ap Extend (g (map f xs)) (map-fusion g f xs)

convert-inv : convert o convert = (λ x → x)
convert-inv = map conv1 o map conv1
= (λ x → (map-fusion conv1 conv1))
map (conv1 o conv1)
= (λ x → (map-idfunc x))
map (λ x → x)
= (λ x → (map-idfunc x))

convert-bijection : Bijection DB DB
convert-bijection = (convert, cast-is-bijection There swap)

Homotopy Type Theory

swapf : (Nat x Nat) → (Nat x Nat)
swapf (x, y) = (y, x)

swap : Bijection (Nat x Nat) (Nat x Nat)
swap = (swapf,
  is-bijection swapf (λ _ → id) (λ _ → id))

There : Type → Type
There A = List (Nat x String x A x Nat)

convert : DB → DB
convert = cast There swap

convert-bijection : Bijection DB DB
convert-bijection = (convert, cast-is-bijection There swap)

Writes proofs for you!
Canonicity for 2-Dimensional Type Theory
POPL’12
More applications

- For modular code, can reason about a fast implementation using a reference implementation: \( \text{cast} \) a proof about the reference implementation to the fast implementation

- Can program domain-specific program verification logics, using \( \text{cast} \) to implement the structural properties [thesis + MFPS’11]
Conclusion
Types are $\infty$-groupoids

```plaintext
<program> : <type>
<proof> : <prog_1> = <prog_2>
<2-proof> : <proof_1> = <proof_2>
<3-proof> : <2-proof_1> = <2-proof_2>
::
```

Proofs, 2-proofs, 3-proofs, ... all influence how a program runs
Homotopy Type Theory

new programs like cast

category theory

type theory

new computer-checked proofs

homotopy theory
Papers and code

1. Fundamental group of the circle [LICS’13]
   Formality homotopy: github.com/dlicata335/

2. Computational interpretation
   of 2D type theory [POPL’12]

3. Domain-specific program verification logics
   [thesis+MFPS’11]

4. The HoTT Book (coming soon!): doing math
   informally in Homotopy Type Theory

5. Blog: homotopytypetheory.org
Research Agenda

- Develop a computational interpretation for infinite-dimensional types (in progress)
- Implement a new proof assistant based on it
- Computer-checked math, especially in category theory and homotopy theory
- Computer-checked software
Parallelism and Verification

**Goal:** fast parallel implementation, proved correct relative to list implementation, in a proof assistant!
Research Agenda

Make it easier to use proof assistants to develop math and software

- PL: languages for expressing mathematics
- SE: managing large codebases
- Compilers + distributed computing: speed
- Machine learning: automated proof search
- HCI: usable by “working mathematicians”
- Graphics: visualization
I am developing a computational theory of ∞-groupoids and applying it to computer-checked math and software