Programming and Proving in Homotopy Type Theory

Daniel R. Licata

Carnegie Mellon University &
Institute for Advanced Study


LUXURY
Sunkist Oranges
REG. U.S. PAT. OFF.
GROWN IN U.S.A.
CALIFORNIA
FRUITGROWERS
EXCHANGE
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

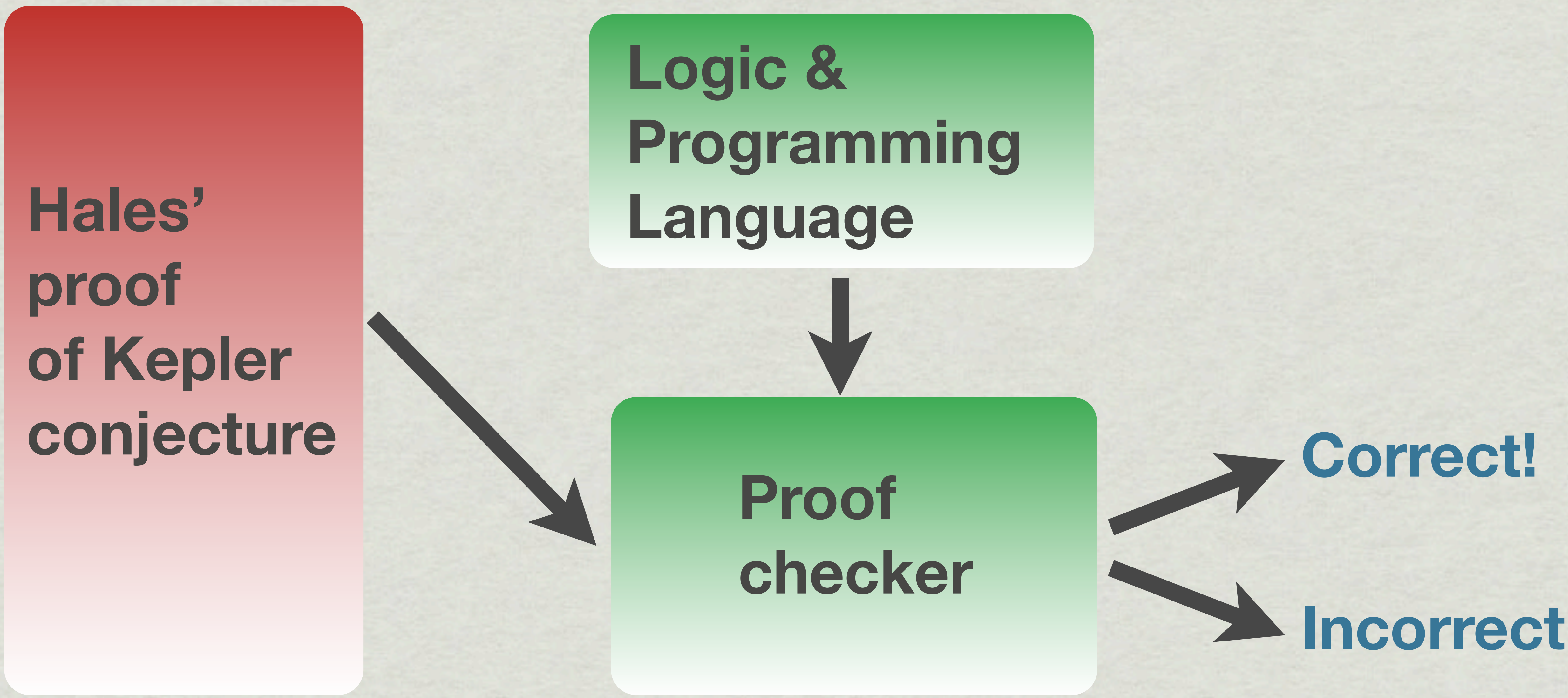
In 2003, after 4 years' work,
12 referees had checked lots of lemmas,
but gave up on verifying the proof

Proofs can be hard to check

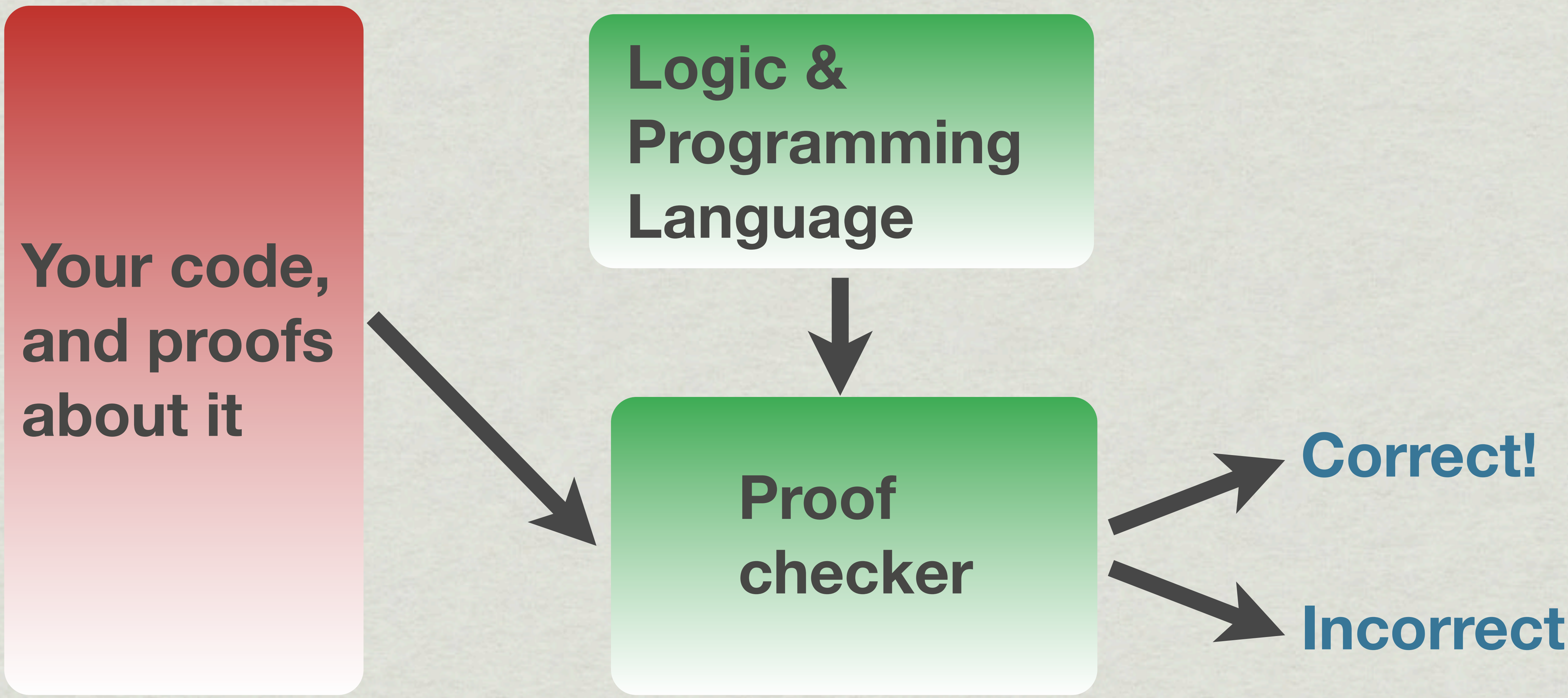
In 2003, after 4 years' work,
12 referees had checked lots of lemmas,
but gave up on verifying the proof

"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



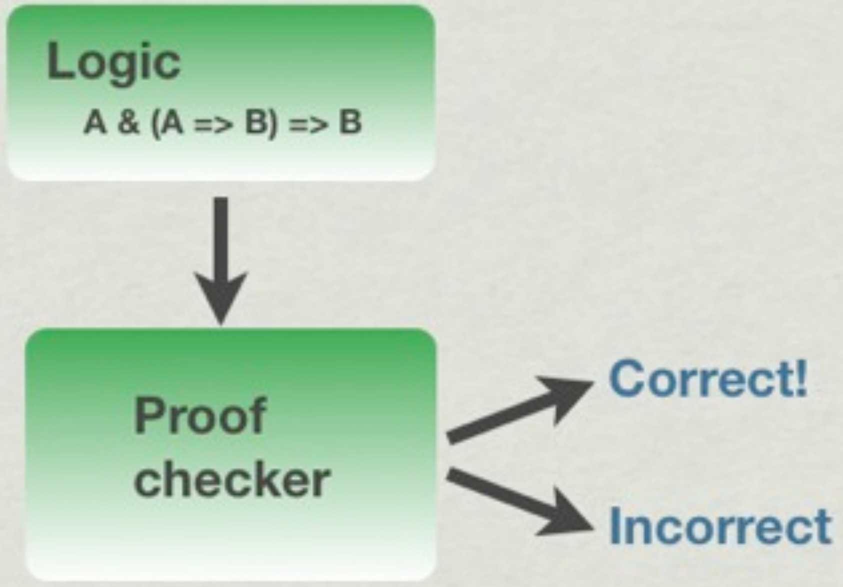
Computer-checked software



Computer-assisted proofs

Proof assistant

- Interactive proof editor
- Automated proofs
- Libraries



Computer-assisted proofs

- are much easier to believe:
  computer does the journal reviewing
- can use computational methods
  and still be fully rigorous
- broaden access:
  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

~5-10x longer

- >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 **~5-10x longer**
- >2 years to run **~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 **~5-10x longer**
- >2 years to run **~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

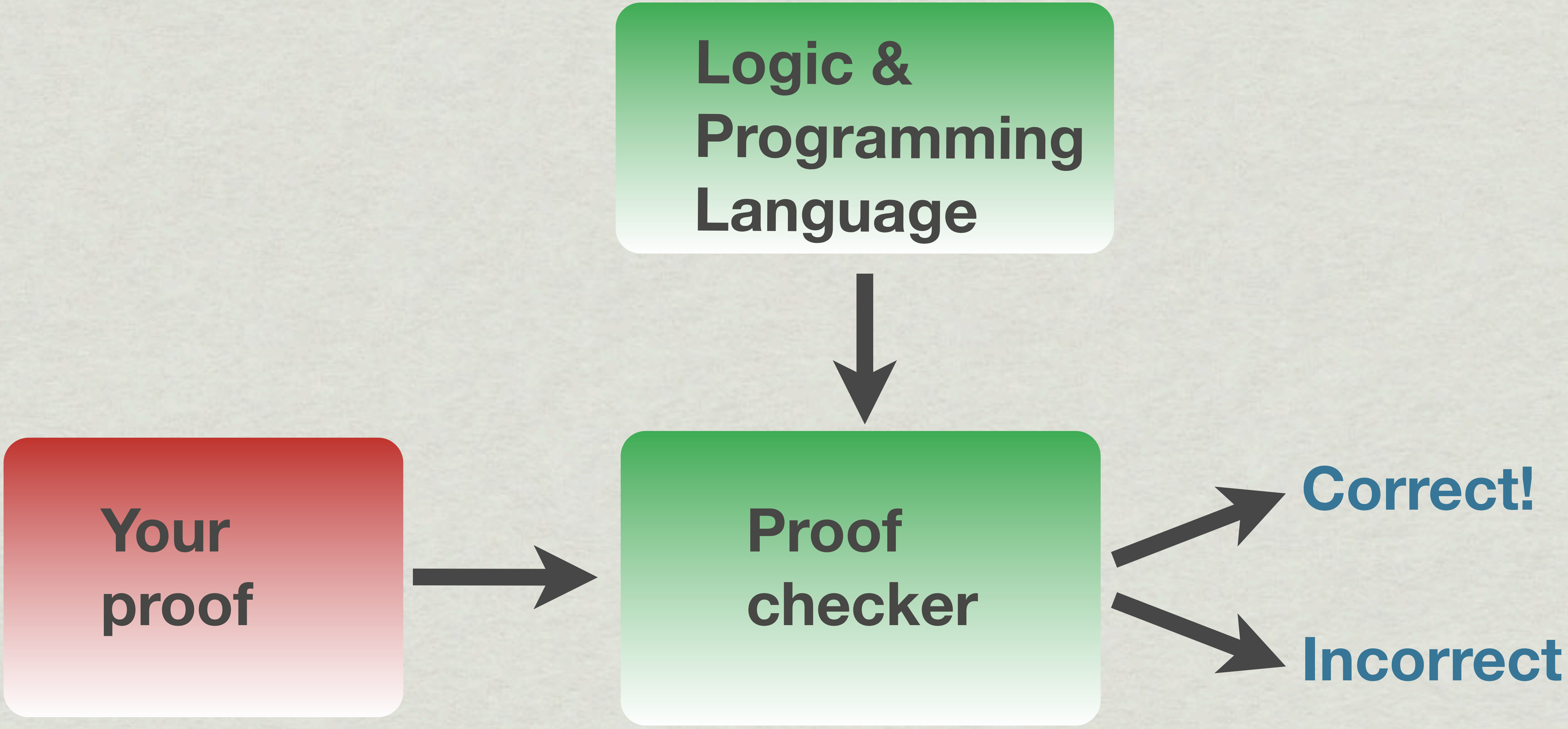
Compilers + distributed computing: speed

Machine learning: automated proof search

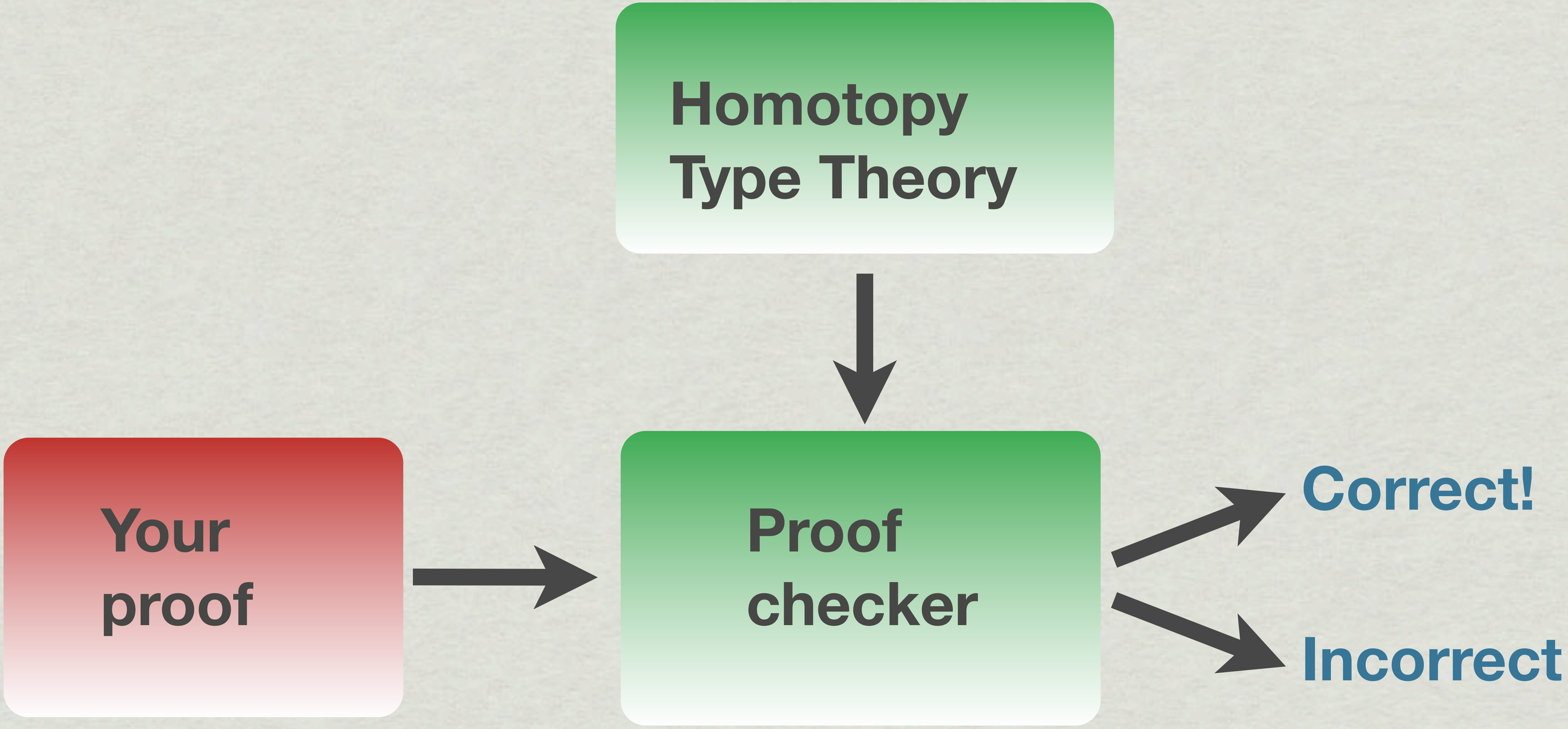
HCI: usable by “working mathematicians”

Graphics: visualization

Homotopy Type Theory



Homotopy Type Theory



Type Theory

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

- Functional programming language

```
insertsort : list<int> → list<int>
mergesort  : list<int> → 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

```
∀x. mergesort(x) = 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**

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)
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	**set theory**
`<program> : <type>`	$x \in S$
`<prog1> = <prog2>`	$x = y$

Type are sets?

Traditional view:

type theory	**set theory**
`<program> : <type>`	`x ∈ S`
`<prog1> = <prog2>`	`x = y`

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?

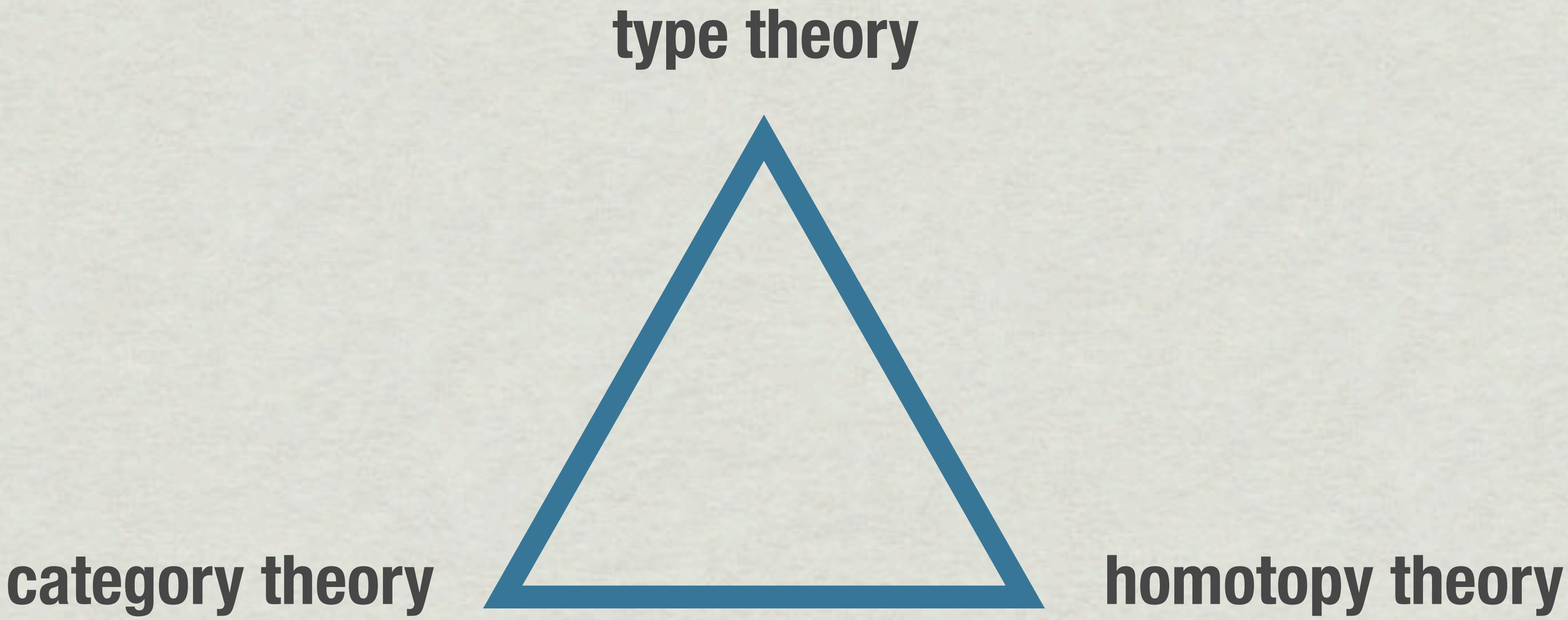
Traditional view:

type theory	set theory
`<program> : <type>`	$x \in S$
`<proof> : <prog1> = <prog2>`	$x = y$

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



Types are ∞-groupoids

[Hofmann,Streicher,Awodey,Warren,Voevodsky
Lumsdaine,Gambino,Garner,van den Berg]

Types are ∞-groupoids

type theory	set theory
<program> : <type>	$x \in S$
<proof> : <prog$_1$> = <prog$_2$>	$x = y$

Types are ∞-groupoids

type theory

<program> : <type>

<proof> : <prog$_1$> = <prog$_2$>

<2-proof> : <proof$_1$> = <proof$_2$>

set theory

$x \in S$

$x = y$

Types are ∞-groupoids

type theory	**set theory**
<program> : <type>	$x \in S$
<proof> : <prog$_1$> = <prog$_2$>	$x = y$
<2-proof> : <proof$_1$> = <proof$_2$>	
<3-proof> : <2-proof$_1$> = <2-proof$_2$>	

Types are ∞-groupoids

type theory

<program> : <type>

<proof> : <prog$_1$> = <prog$_2$>

<2-proof> : <proof$_1$> = <proof$_2$>

<3-proof> : <2-proof$_1$> = <2-proof$_2$>

⋮

set theory

$x \in S$

$x = y$

Types are ∞-groupoids

type theory

<program> : <type>

<proof> : <prog$_1$> = <prog$_2$>

<2-proof> : <proof$_1$> = <proof$_2$>

<3-proof> : <2-proof$_1$> = <2-proof$_2$>

⋮

set theory

$x \in S$

$x = y$

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

Types are ∞-groupoids

type theory

<program> : <type>

<proof> : <prog$_1$> = <prog$_2$>

<2-proof> : <proof$_1$> = <proof$_2$>

<3-proof> : <2-proof$_1$> = <2-proof$_2$>

⋮

set theory

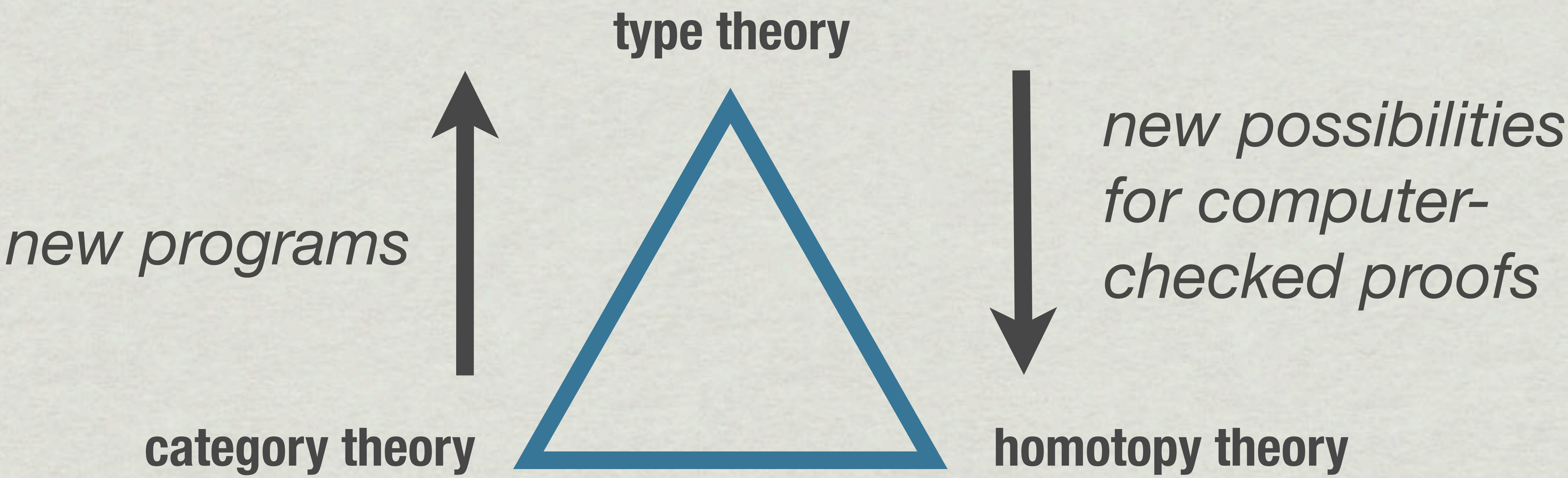
$x \in S$

$x = y$

**∞-*group*oid:
each level has a
group structure,
and they interact**

***Proofs, 2-proofs, 3-proofs, …
all influence how a program runs***

Homotopy Type Theory



I am developing
a computational theory of ∞-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

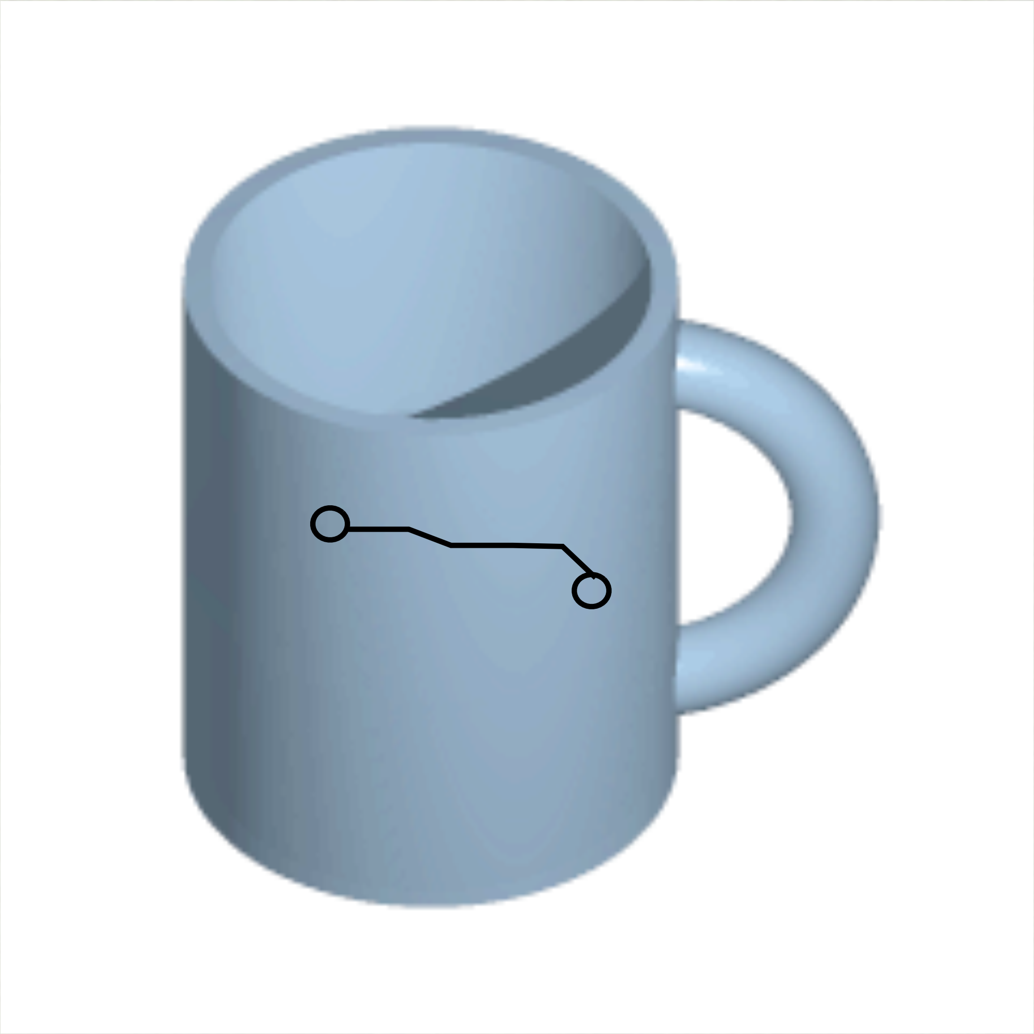
Homotopy Theory

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

[image from wikipedia]

Homotopy Theory

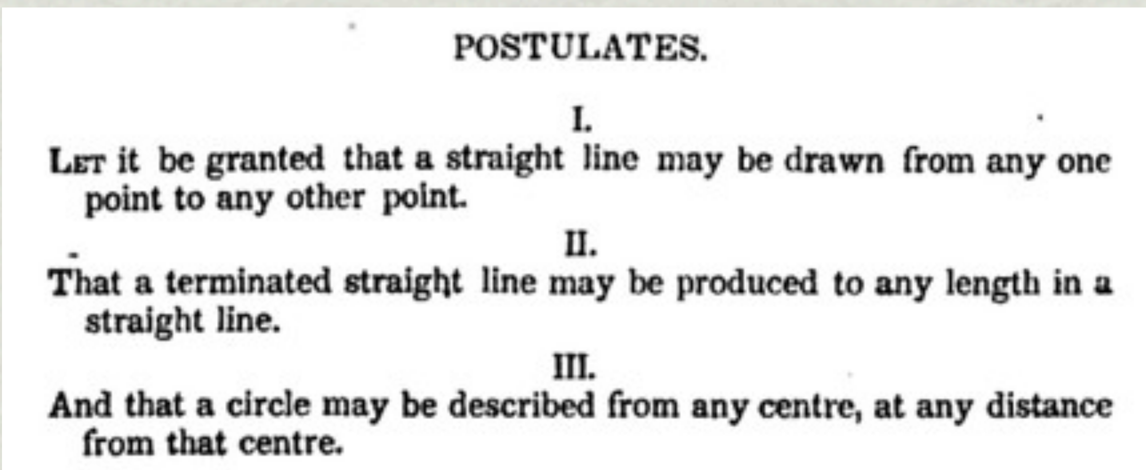
A branch of topology,
the study of spaces and continuous deformations



[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.

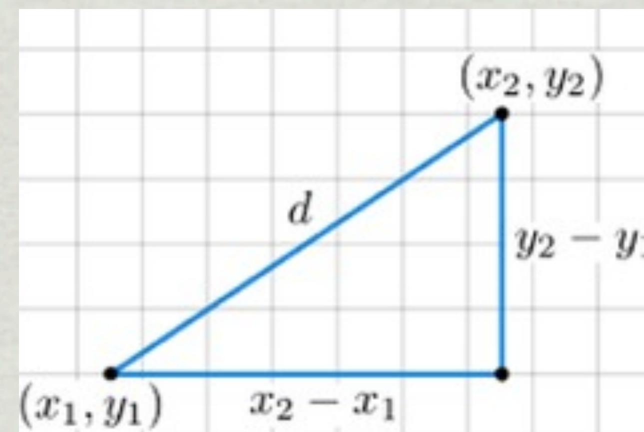
II.

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.

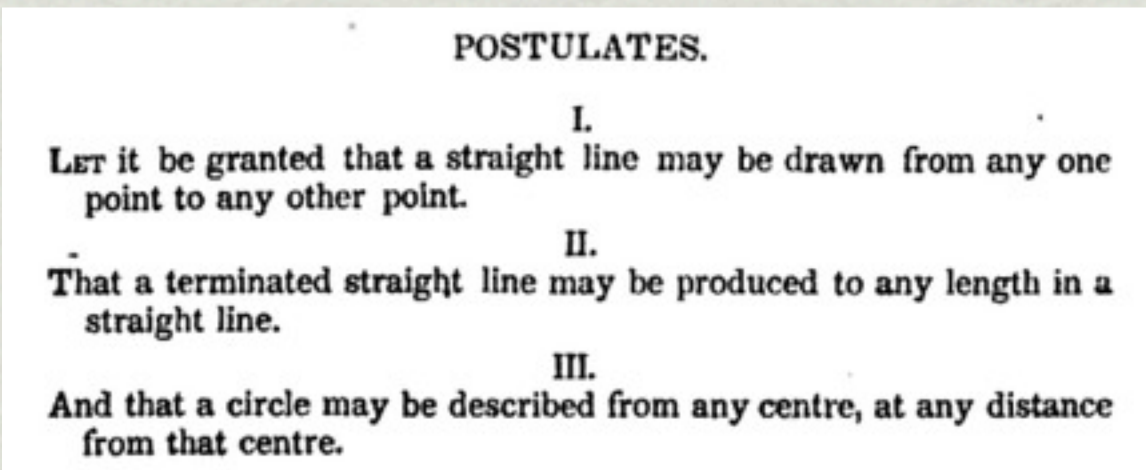
Analytic geometry (Descartes)



[image from wikipedia]

Synthetic vs Analytic

Synthetic geometry (Euclid)


POSTULATES.

I.

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

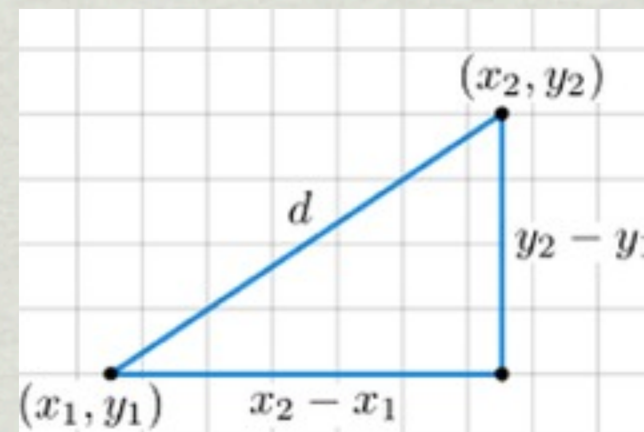
II.

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)



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

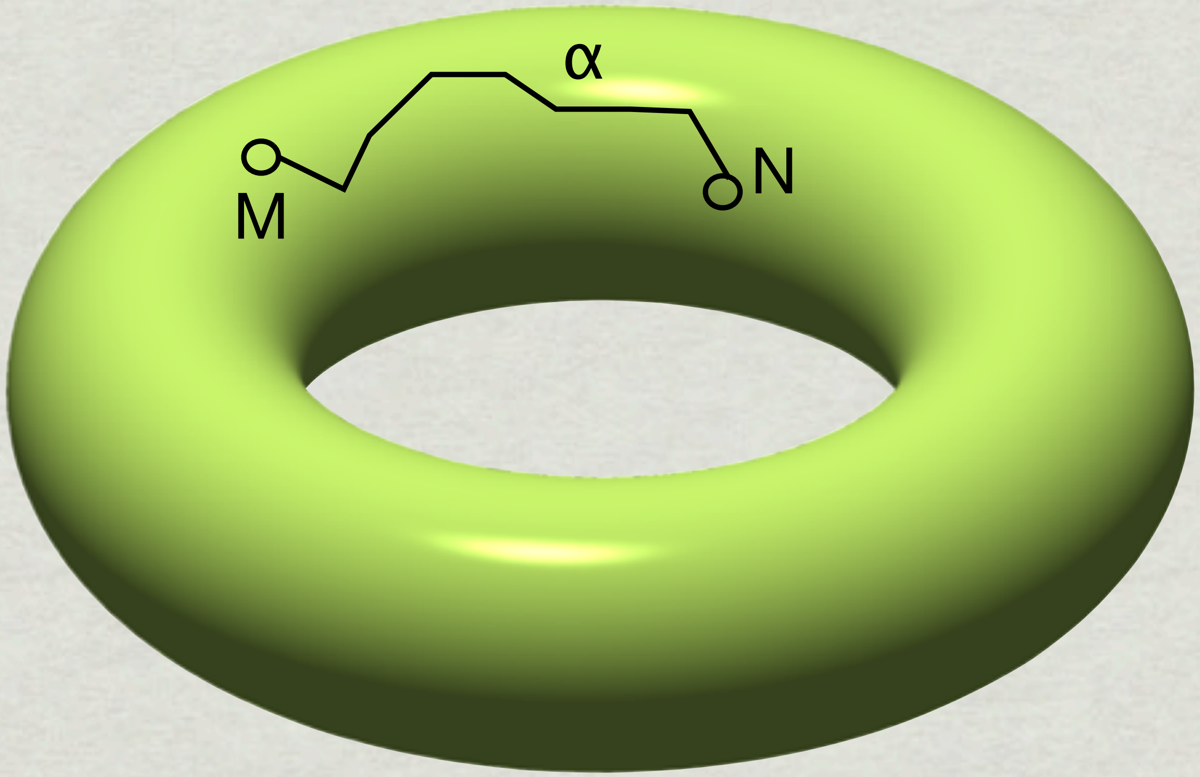
homotopy theory	**type theory**
space	<type>
points	<program> : <type>
paths	<proof> : <prog$_1$> = <prog$_2$>
homotopies	<2-proof> : <proof$_1$> = <proof$_2$>
⋮	⋮

Synthetic homotopy theory

homotopy theory	**type theory**
space	<type>
points	<program> : <type>
paths	<proof> : <prog$_1$> = <prog$_2$>
homotopies	<2-proof> : <proof$_1$> = <proof$_2$>
⋮	⋮

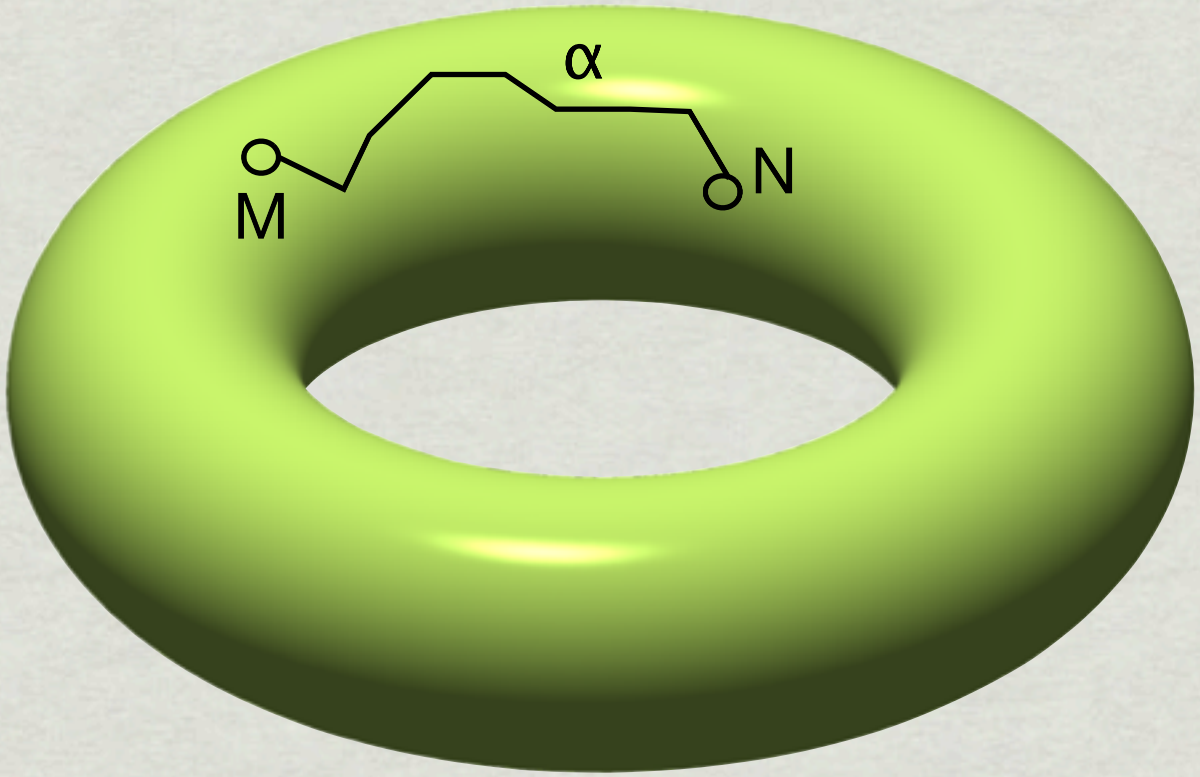
*A path is **not** a set of points; it is a primitive notion*

Spaces as types



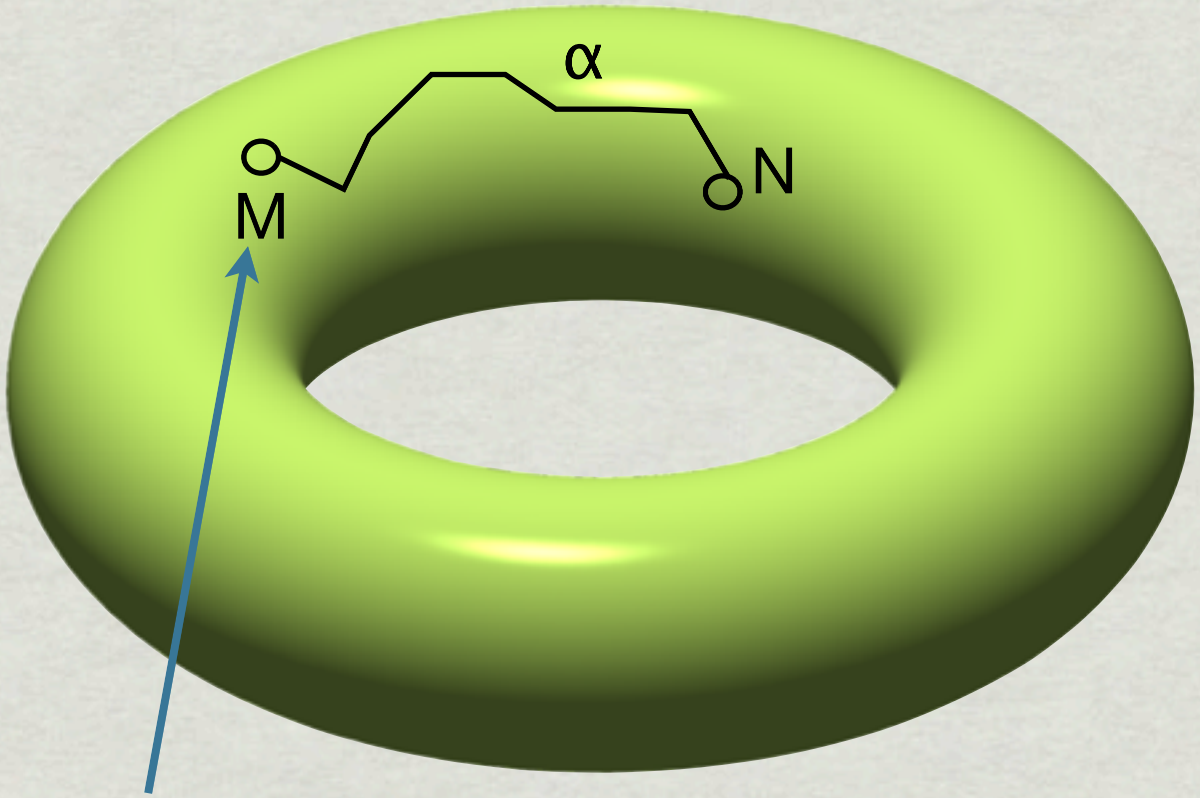
Spaces as types

a space is a type A



Spaces as types

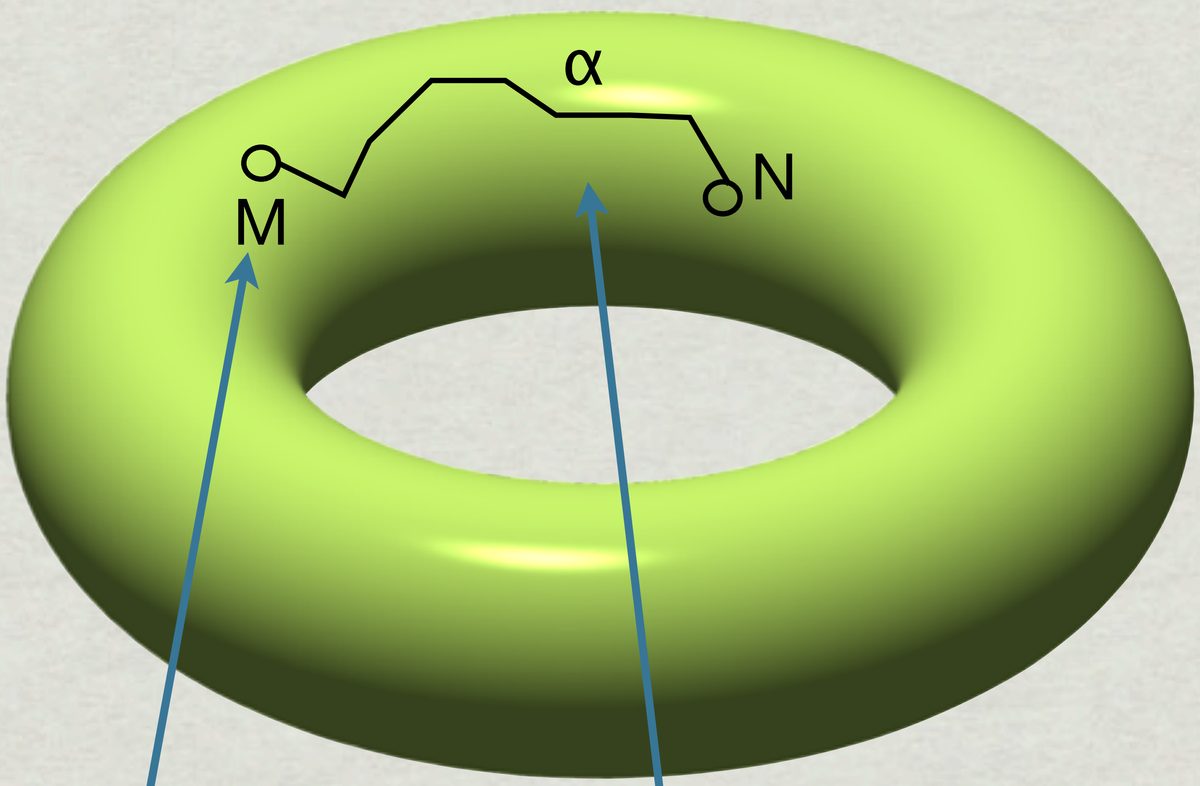
a space is a type A



points are programs
M : A

Spaces as types

a space is a type A



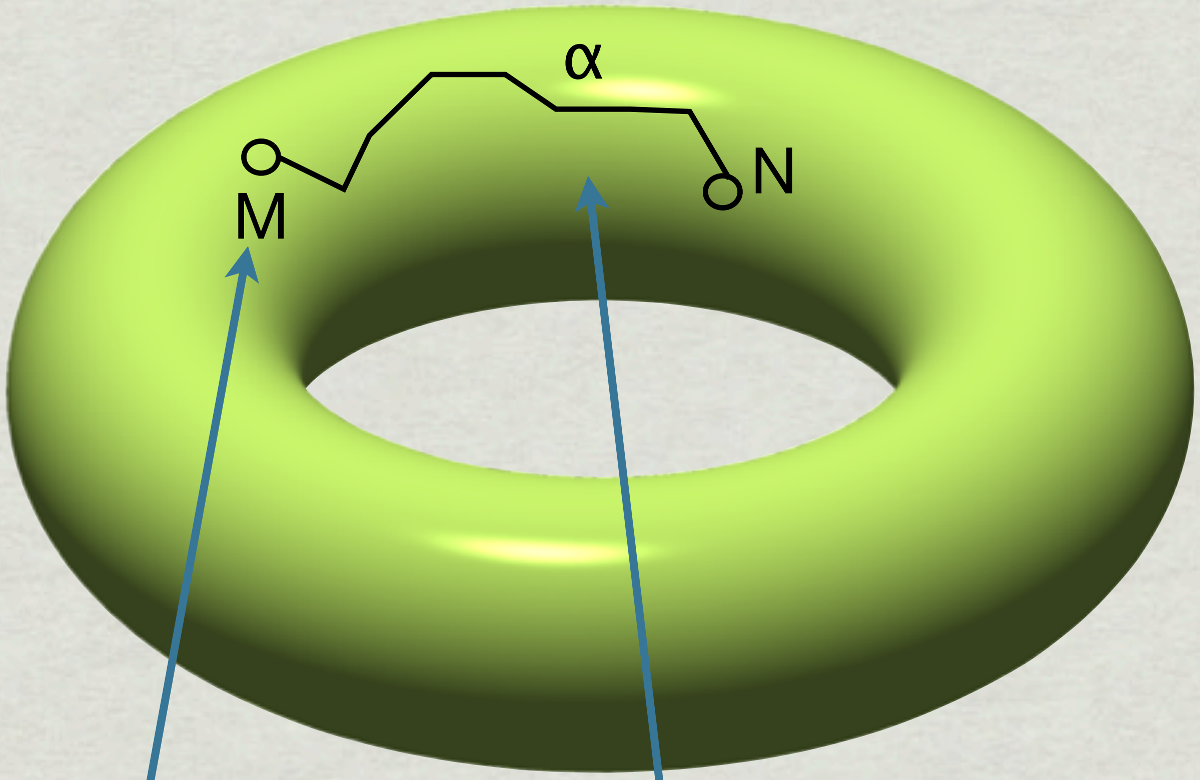
points are programs
$M : A$

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

Spaces as types

a space is a type A

path operations

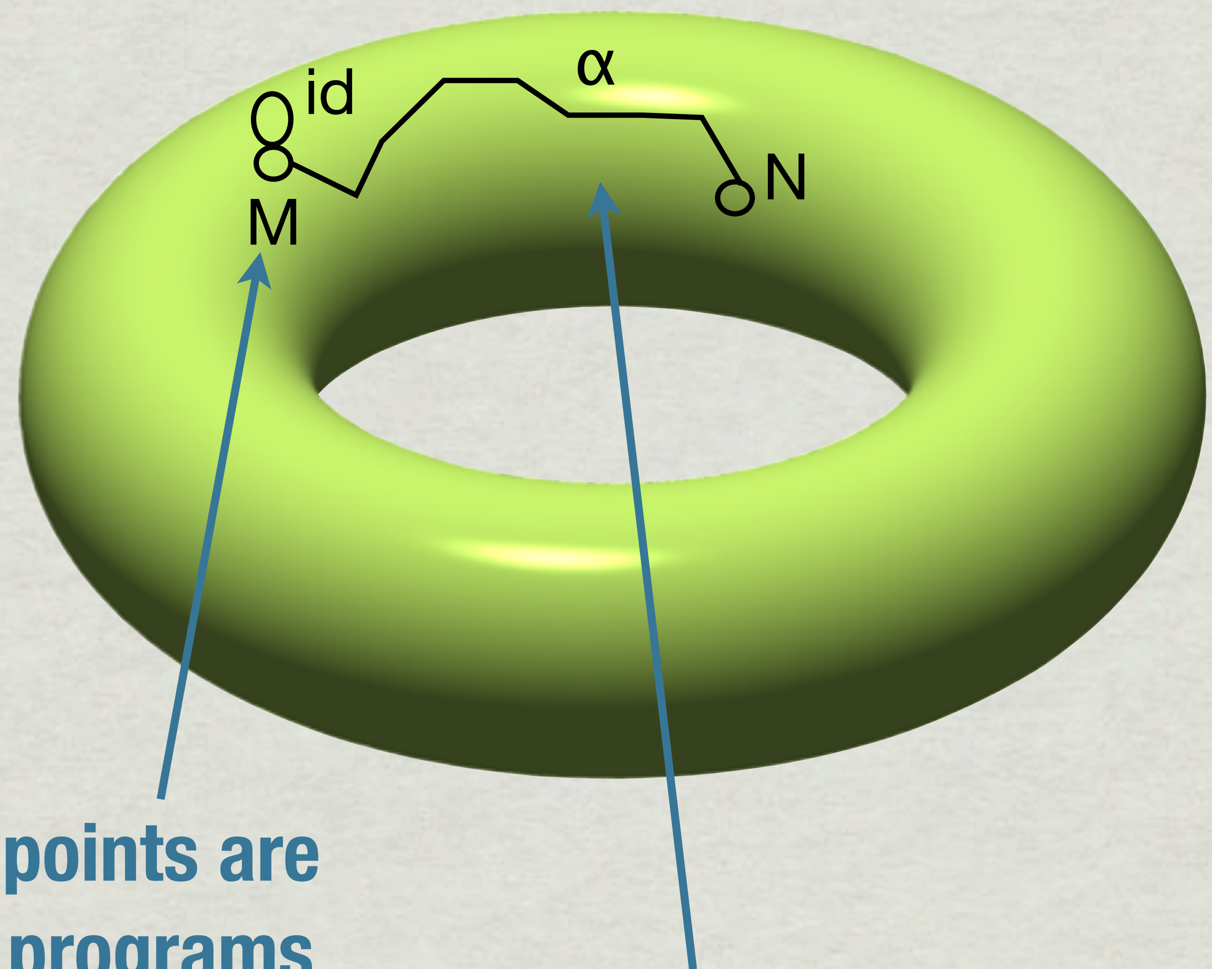


points are programs
M : A

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

Spaces as types

a space is a type A



points are programs
M : A

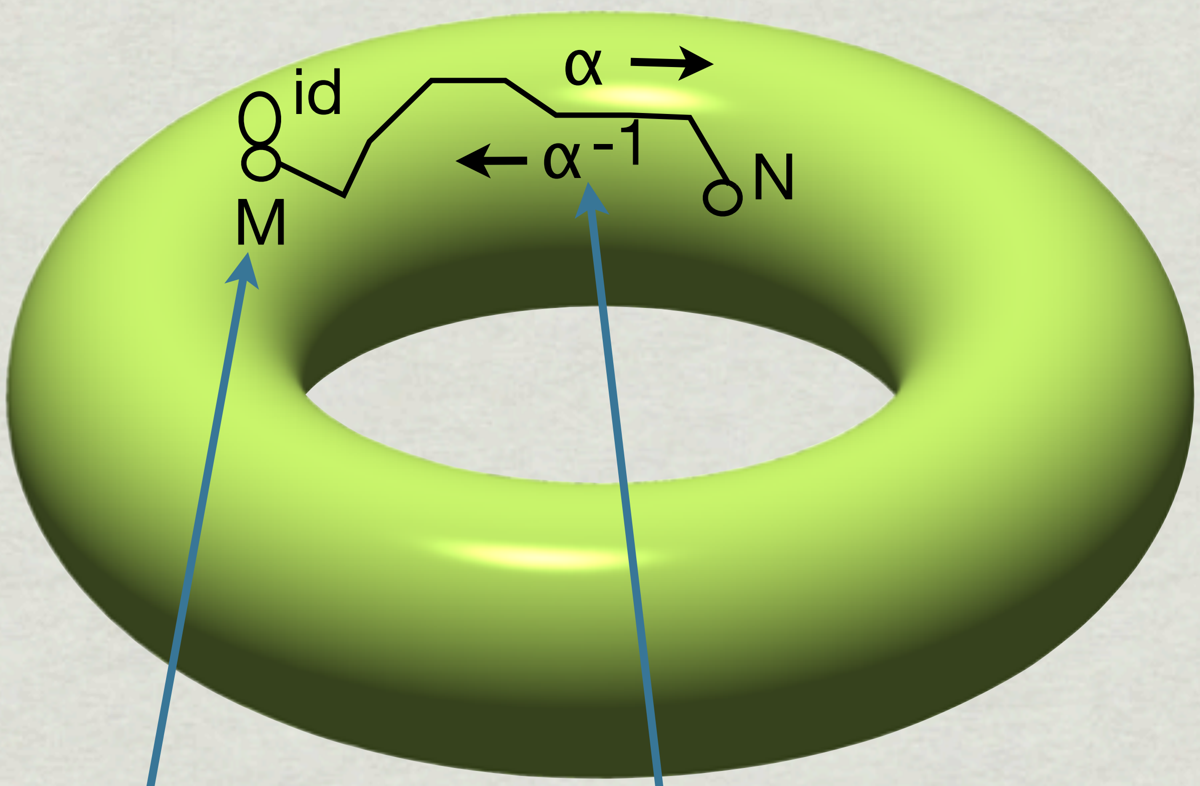
paths are *proofs of equality*
$\alpha : M =_A N$

path operations

```
id      : M = M (refl)
```

Spaces as types

a space is a type A



points are programs
M:A

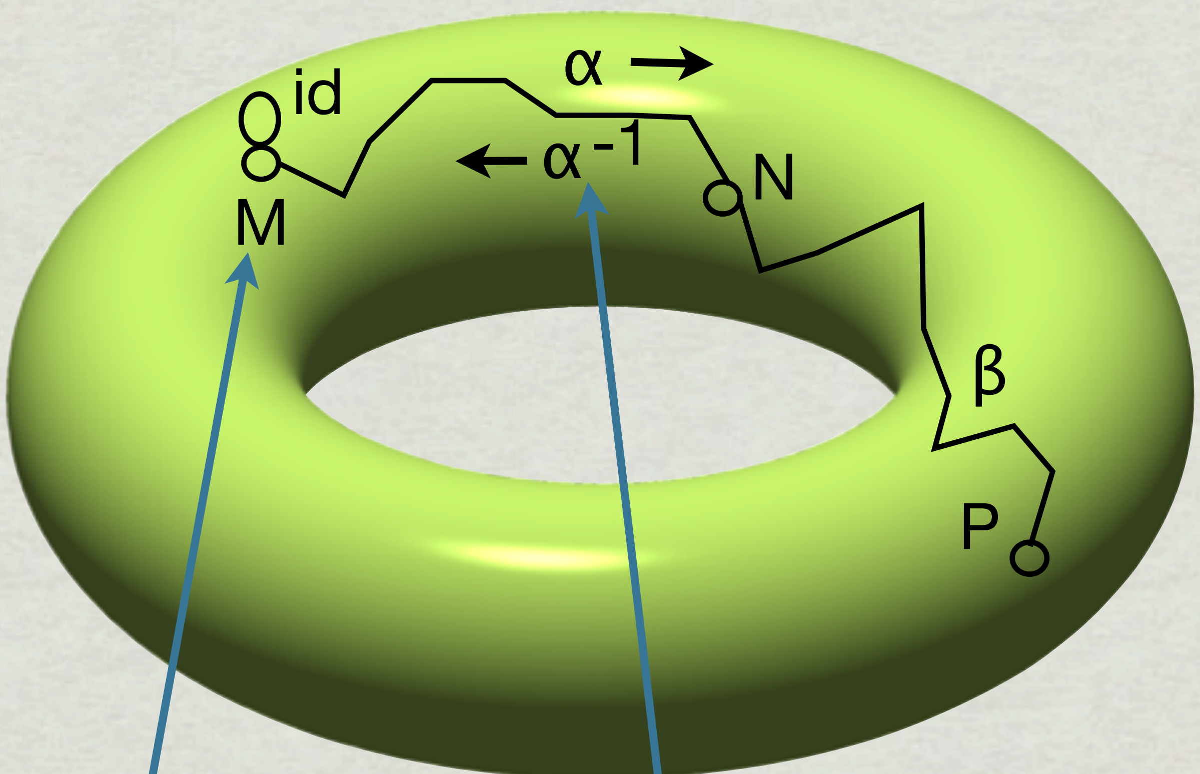
paths are
proofs of equality
$\alpha : M =_A N$

path operations

```
id     : M = M (refl)
α^-1   : N = M (sym)
```

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

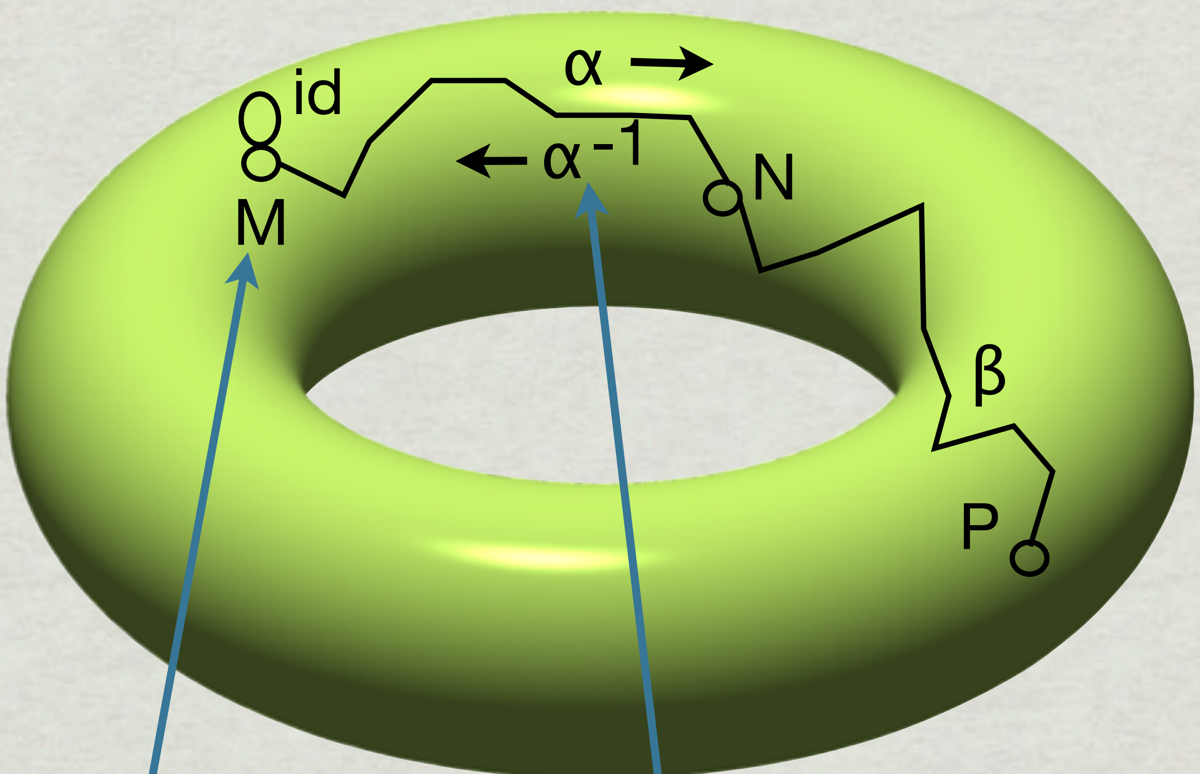
$\text{id} : M = M$ (refl)

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

$\beta \circ \alpha : M = P$ (trans)

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)

α^{-1} : N = M (sym)

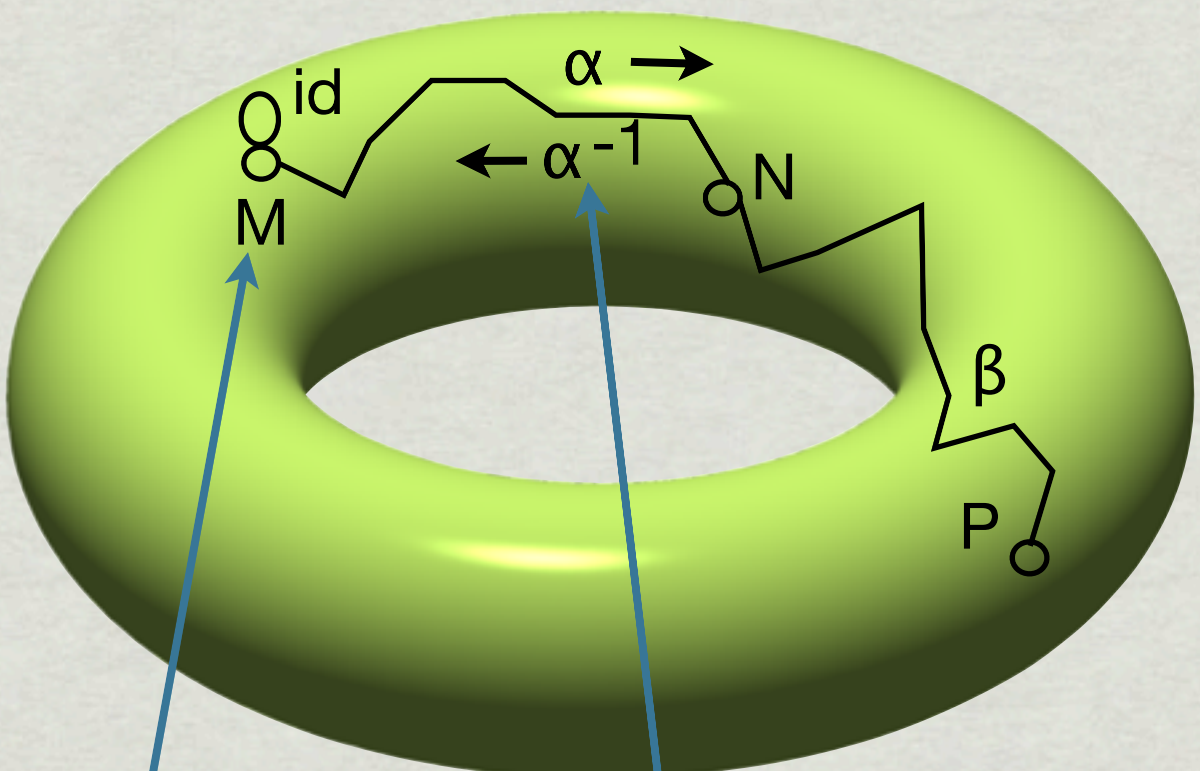
β o α : M = P (trans)

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

id	: M = M	(refl)
α^{-1}	: N = M	(sym)
β o α	: M = P	(trans)

Fundamental group:

group of loops

modulo homotopy

Homotopy

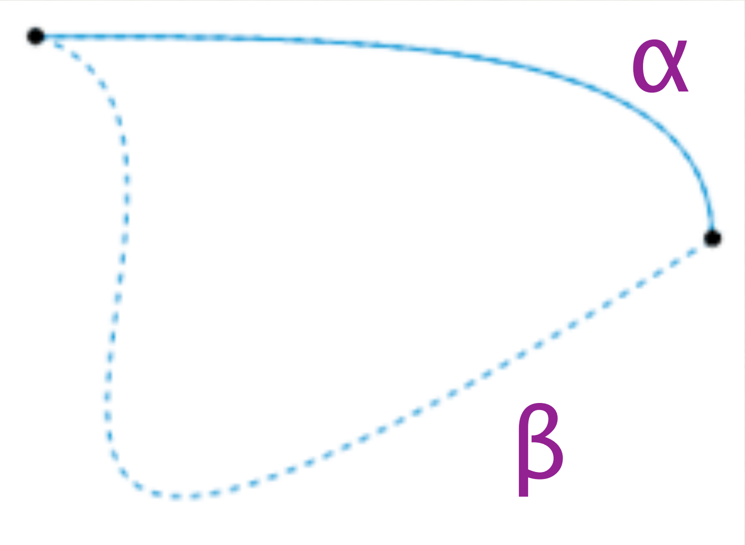
Deformation of one path into another

α

β

Homotopy

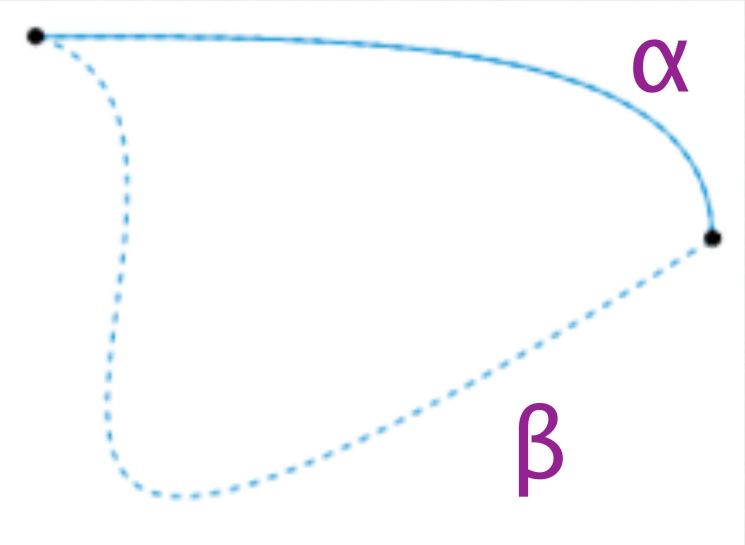
Deformation of one path into another



[image from wikipedia]

Homotopy

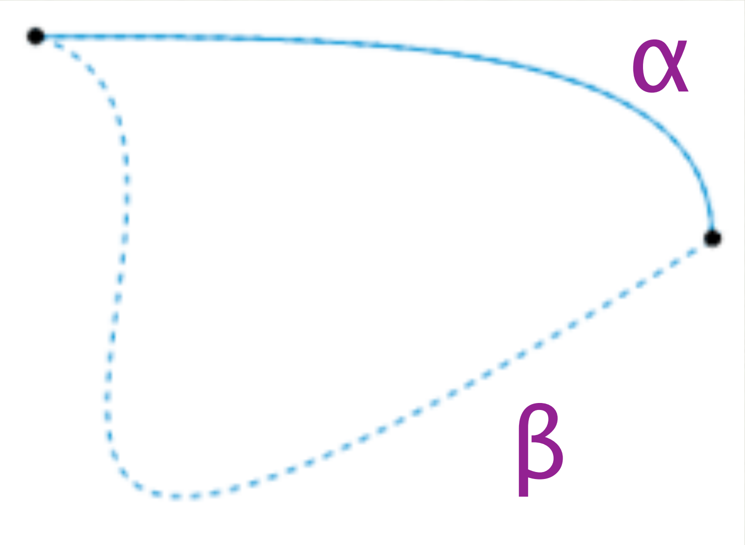
Deformation of one path into another



[image from wikipedia]

Homotopy

Deformation of one path into another

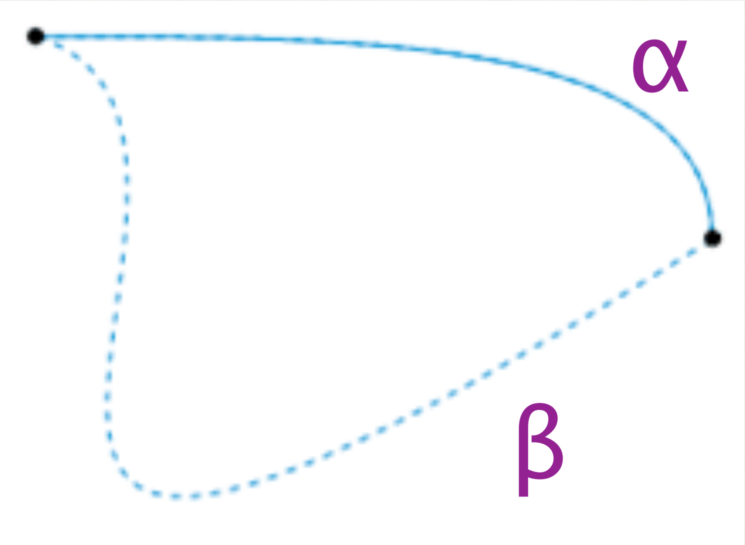


= 2-dimensional *path between paths*

[image from wikipedia]

Homotopy

Deformation of one path into another



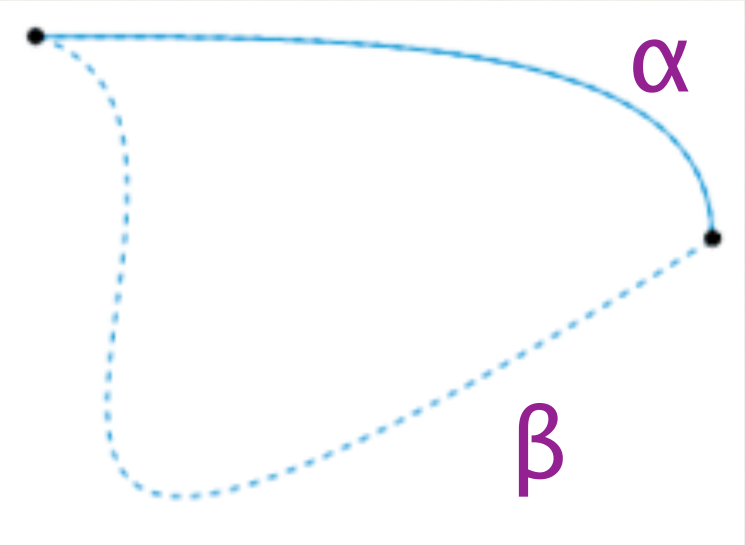
`<2-proof> : α = β`

= 2-dimensional *path between paths*

[image from wikipedia]

Homotopy

Deformation of one path into another



<2-proof> : α = β

= 2-dimensional *path between paths*

Homotopy theory is the study of spaces by way of their paths, homotopies, homotopies between homotopies, ….

[image from wikipedia]

We can do homotopy theory by writing functional programs

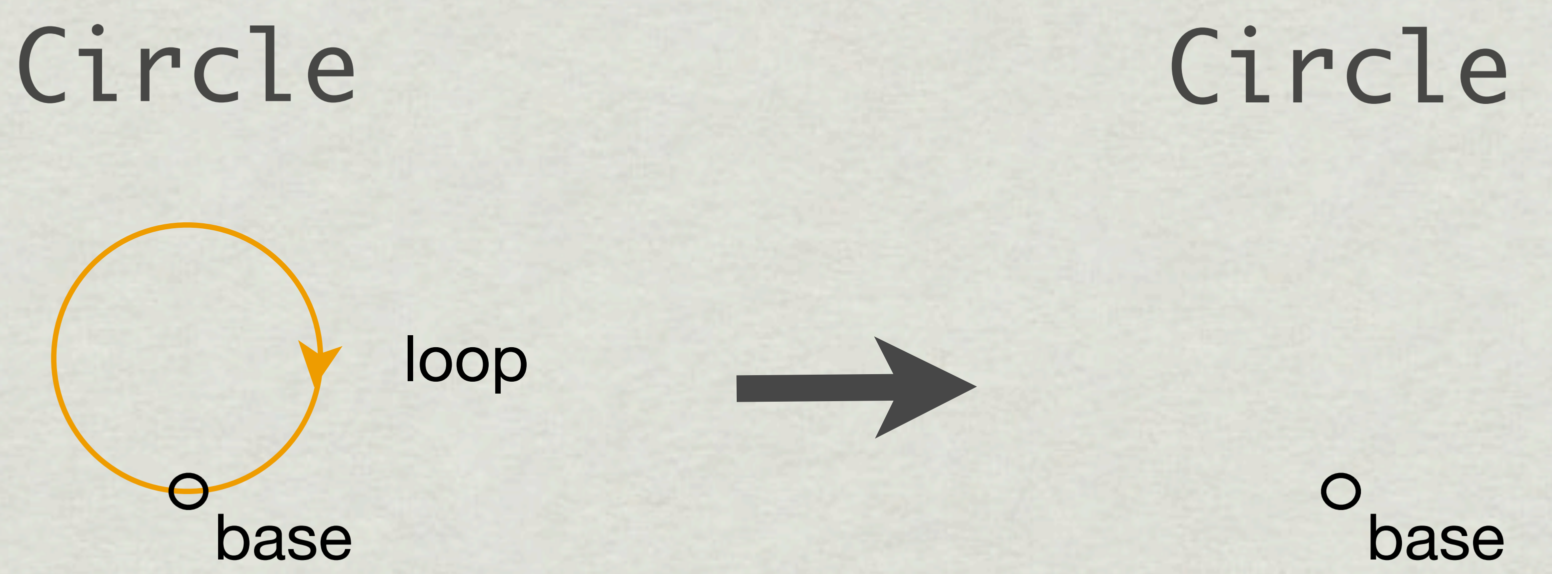
Functions on sets

Function on a set gives the image of each element:

```
not : Bool → Bool
not(true) = false
not(false) = true
```

Functions on spaces

Function on a space gives the image of each point



Functions on spaces

Function on a space gives the image of each point

and each path!

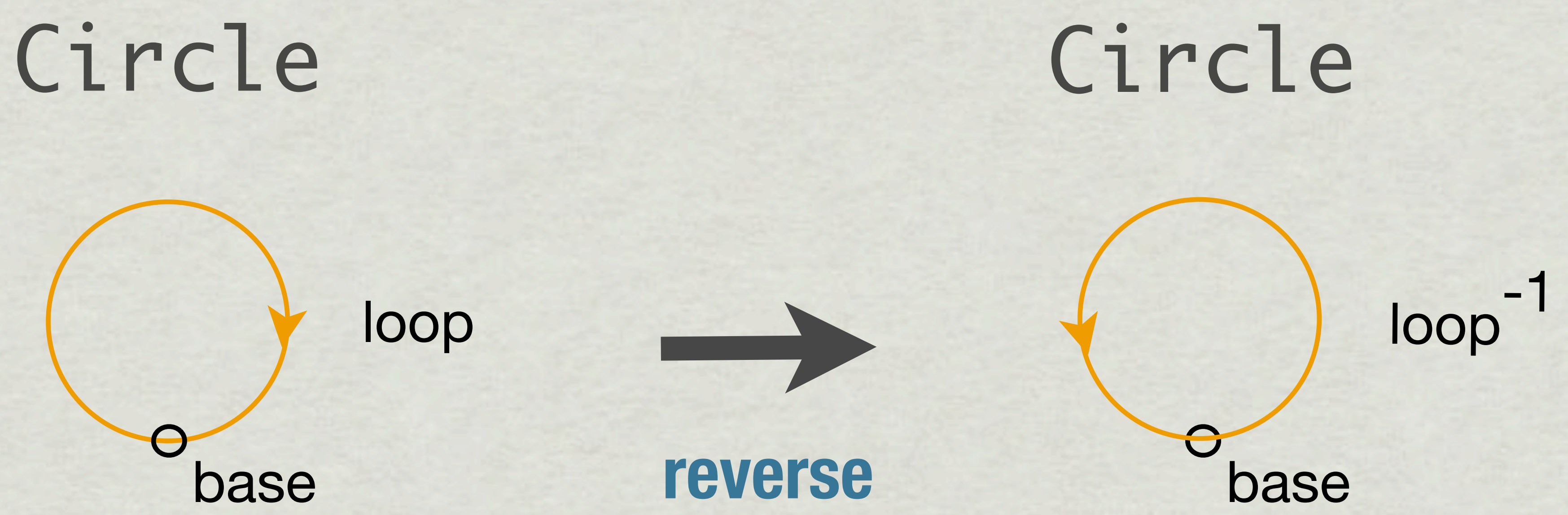
Functions on spaces

Function on a space gives the image of each point

and each path!

Functions on spaces

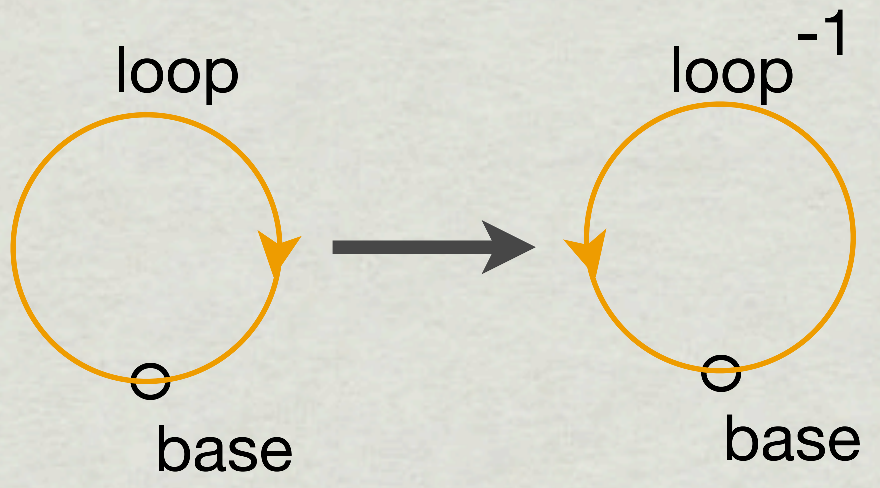
Function on a space gives the image of each point



and each path!

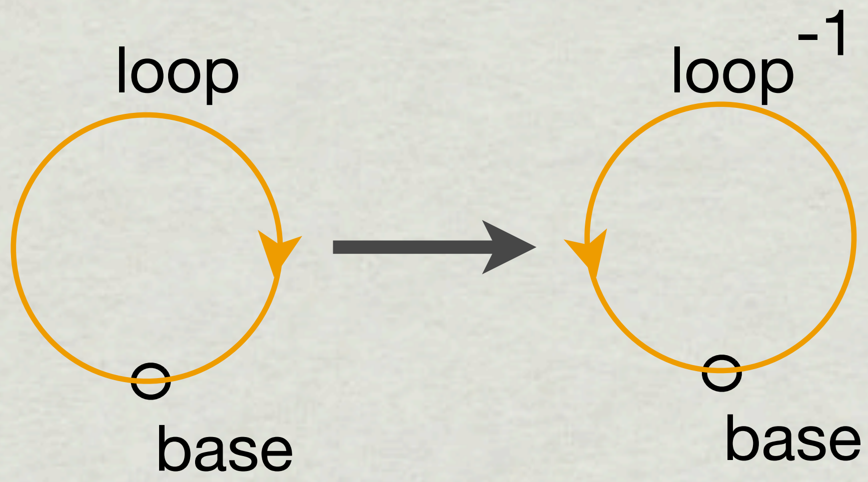
Circle Recursion

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```



Circle Recursion

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```

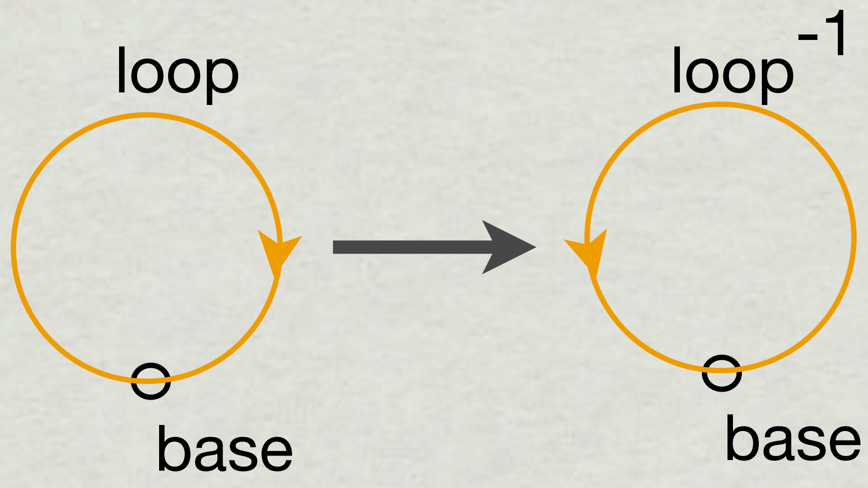


This specifies the image for **all paths** because

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

Homomorphism

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```

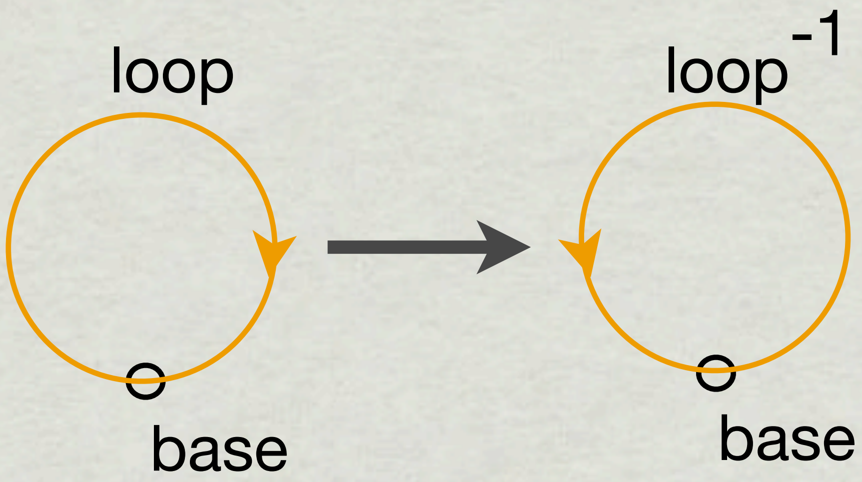


Computation steps:

```
    reverse(loop o loop)
```

Homomorphism

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```

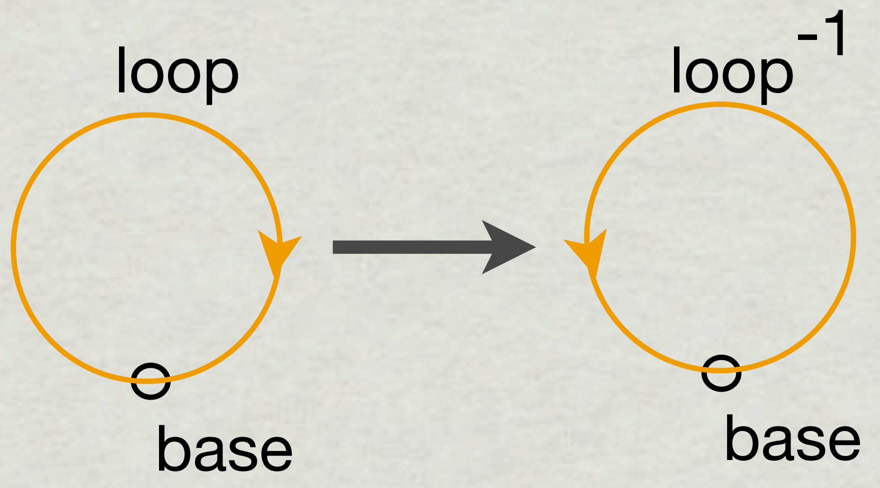


Computation steps:

```
   reverse(loop o loop)
 = (reverse loop) o (reverse loop)
```
homomorphism

Homomorphism

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```

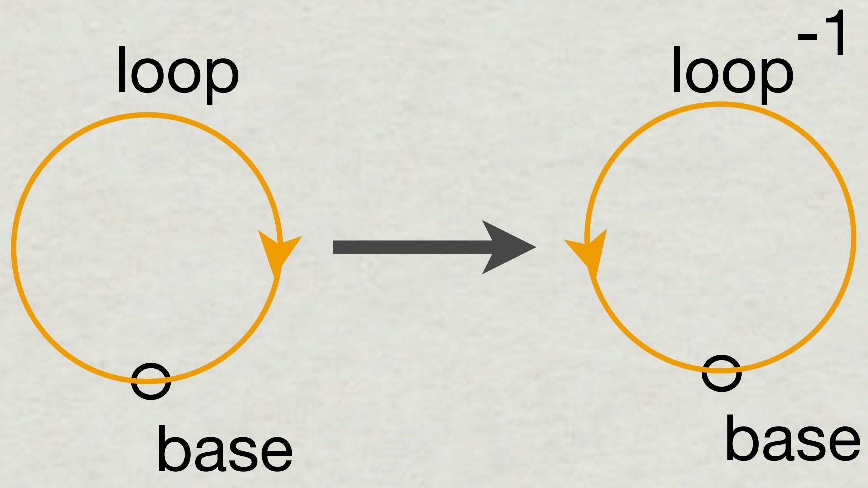


Computation steps:

```
   reverse(loop o loop)
 = (reverse loop) o (reverse loop)   homomorphism
 = loop⁻¹ o loop⁻¹                   definition
```

Homomorphism

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```



Computation steps:

reverse(loop o loop)

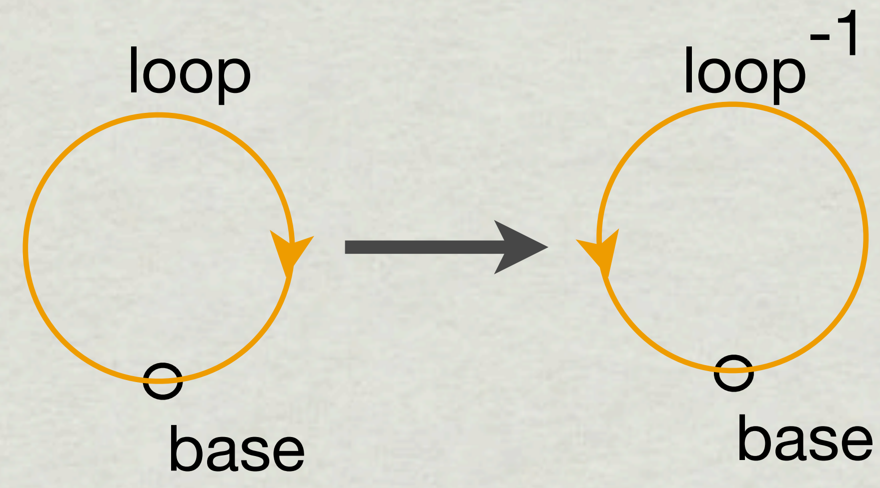
= (reverse loop) o (reverse loop) **homomorphism**

= loop^{-1} o loop^{-1} **definition**

= $(\text{loop o loop})^{-1}$ **group laws**

Homomorphism

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```

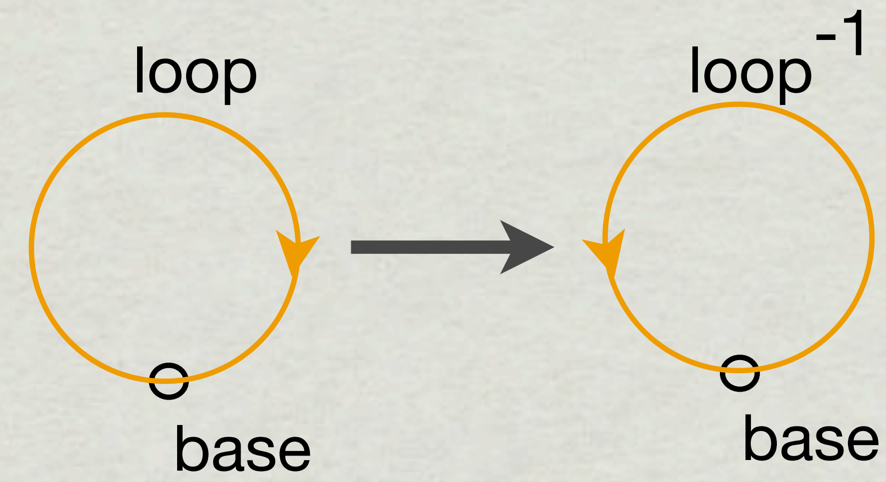


Computation steps:

```
   reverse(loop o loop)
 = (reverse loop) o (reverse loop)   homomorphism
 = loop⁻¹ o loop⁻¹                    definition
 = (loop o loop)⁻¹                    group laws
```

Circle induction

```
reverse : Circle → Circle
reverse(base) = base
reverse(loop) = loop⁻¹
```



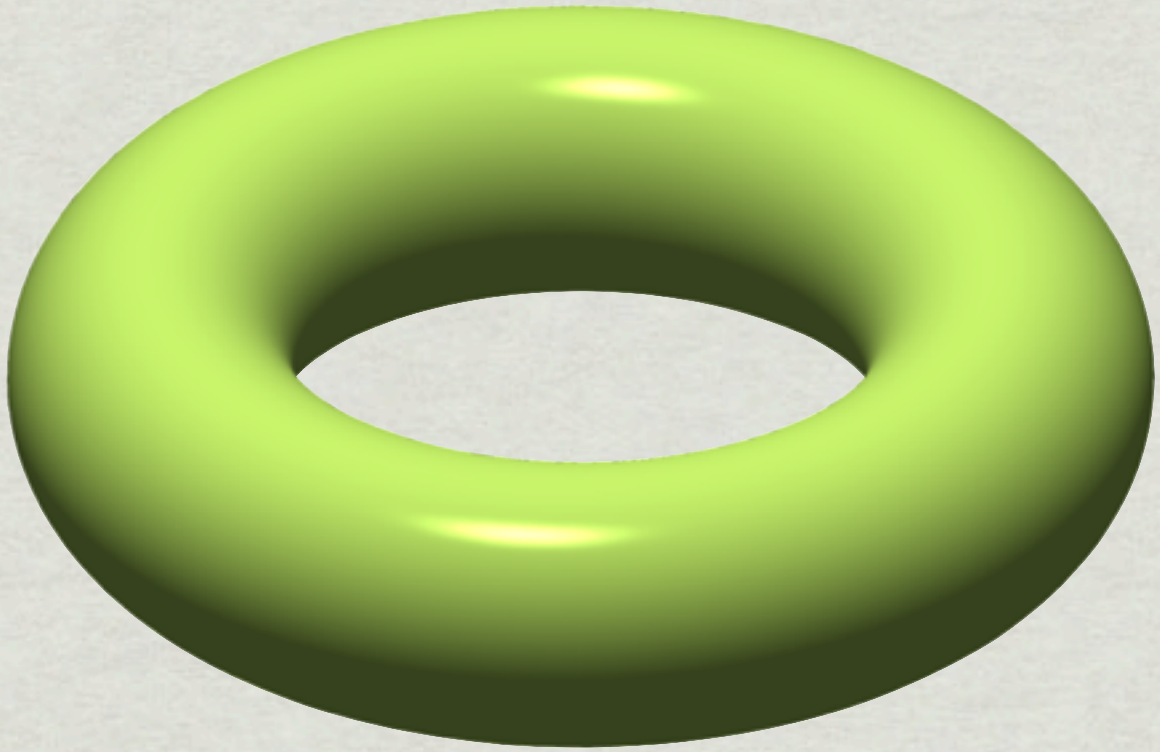
Theorem: `∀p. reverse(p) = p`$^{-1}$

Proof: uses *circle induction:*

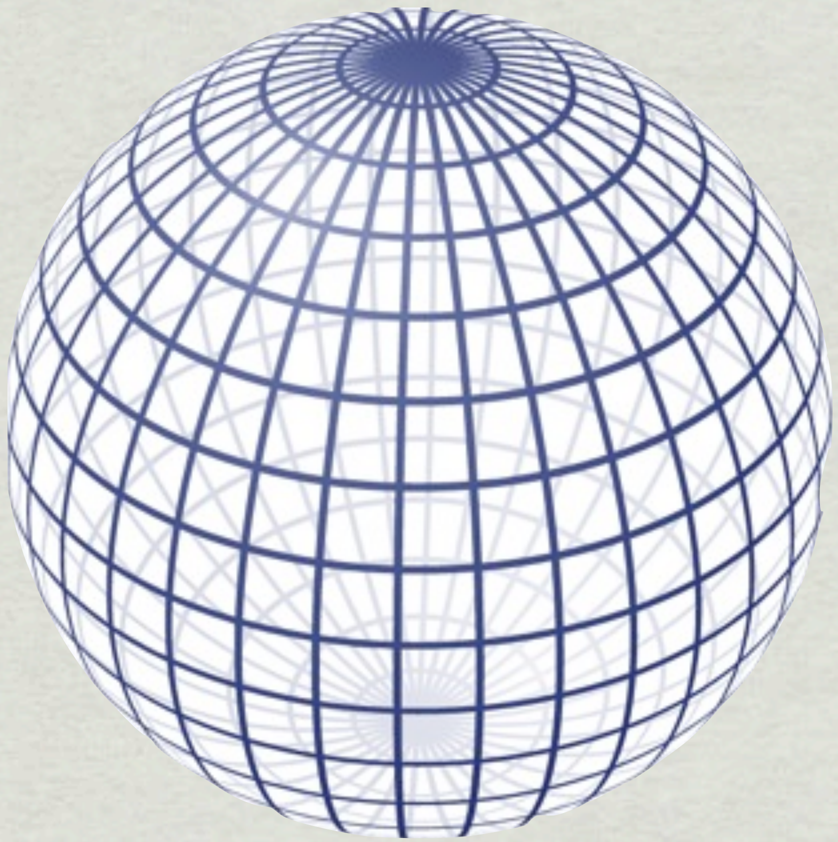
To prove a predicate P for all points on the circle,
suffices to prove P(base),
continuously in the loop

We can do interesting homotopy theory synthetically

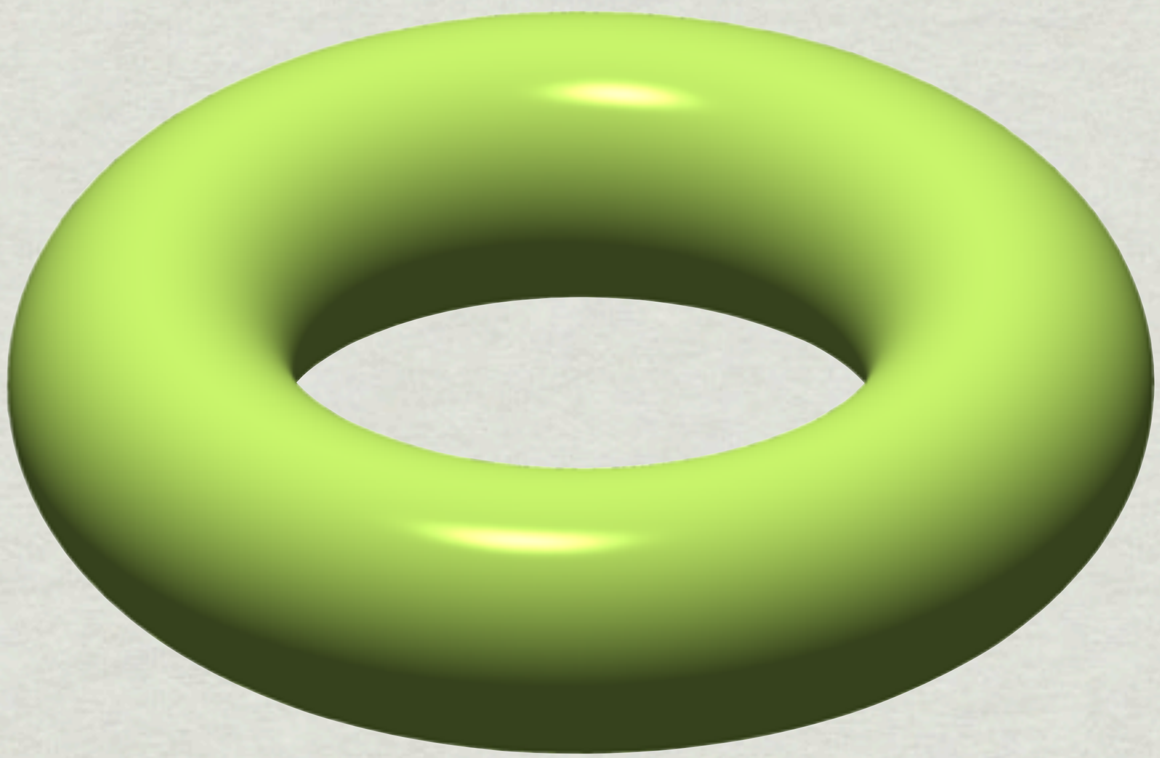
Telling spaces apart



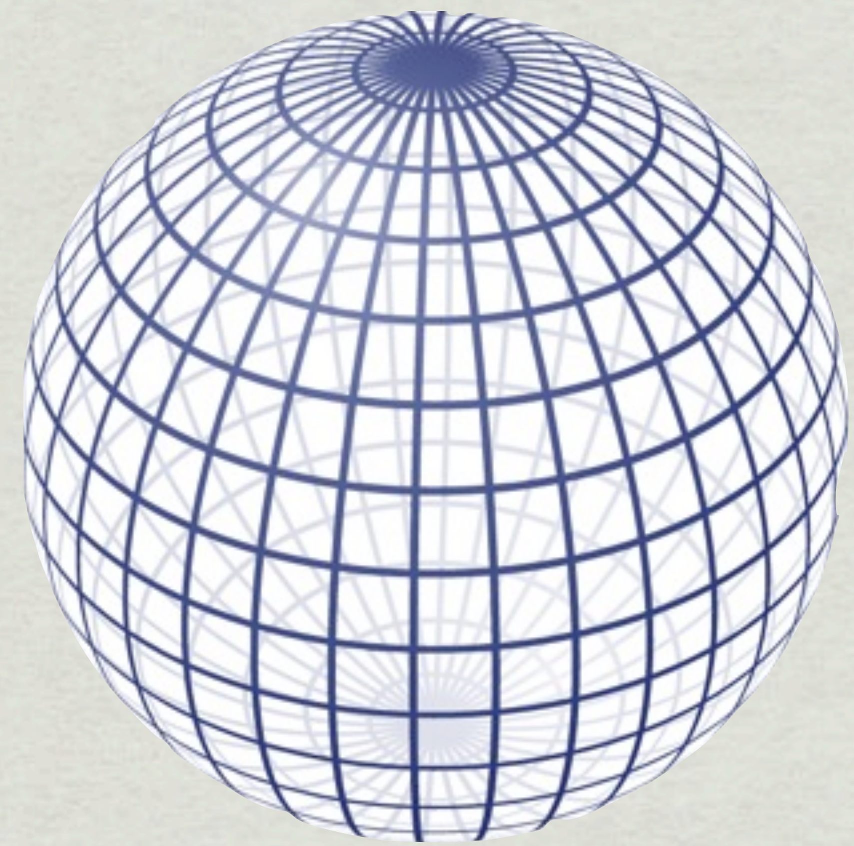
$\neq$



Telling spaces apart



$\neq$



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

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

kth homotopy group

n-dimensional sphere

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	$\mathbf{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	$\mathbf{Z}$	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{12}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$
S^3	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{12}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$
S^4	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}\times\mathbf{Z}_{12}$	$\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{24}\times\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^3$	$\mathbf{Z}_{120}\times\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^5$
S^5	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{30}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^3$	$\mathbf{Z}_{72}\times\mathbf{Z}_2$
S^6	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_{60}$	$\mathbf{Z}_{24}\times\mathbf{Z}_2$	$\mathbf{Z}_2^3$
S^7	0	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	0	$\mathbf{Z}_2$	$\mathbf{Z}_{120}$	$\mathbf{Z}_2^3$
S^8	0	0	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	0	$\mathbf{Z}_2$	$\mathbf{Z}\times\mathbf{Z}_{120}$

[image from wikipedia]

Computer-checked proofs

k^th homotopy group

n-dimensional sphere

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	Z	Z	Z_2	Z_2	Z_{12}	Z_2	Z_2	Z_3	Z_{15}	Z_2	Z_2^2	$Z_{12}{\times}Z_2$	$Z_{84}{\times}Z_2^2$	Z_2^2
S^3	0	0	Z	Z_2	Z_2	Z_{12}	Z_2	Z_2	Z_3	Z_{15}	Z_2	Z_2^2	$Z_{12}{\times}Z_2$	$Z_{84}{\times}Z_2^2$	Z_2^2
S^4	0	0	0	Z	Z_2	Z_2	$Z{\times}Z_{12}$	Z_2^2	Z_2^2	$Z_{24}{\times}Z_3$	Z_{15}	Z_2	Z_2^3	$Z_{120}{\times}Z_{12}{\times}Z_2$	$Z_{84}{\times}Z_2^5$
S^5	0	0	0	0	Z	Z_2	Z_2	Z_{24}	Z_2	Z_2	Z_2	Z_{30}	Z_2	Z_2^3	$Z_{72}{\times}Z_2$
S^6	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	Z	Z_2	Z_{60}	$Z_{24}{\times}Z_2$	Z_2^3
S^7	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0	Z_2	Z_{120}	Z_2^3
S^8	0	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0	Z_2	$Z{\times}Z_{120}$

[image from wikipedia]

Computer-checked proofs

k^{th} homotopy group

n-dimensional sphere

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	Z													
S^3	0	0	Z	Z_2											
S^4	0	0	0	Z	Z_2	Z_2									
S^5	0	0	0	0	Z	Z_2	Z_2	Z_{24}							
S^6	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0					
S^7	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0			
S^8	0	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0	Z_2	

[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 $K(G,n)$

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	$\mathbb{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	$\mathbb{Z}$													
S^3	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$											
S^4	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$									
S^5	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$							
S^6	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0					
S^7	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0			
S^8	0	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	

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”

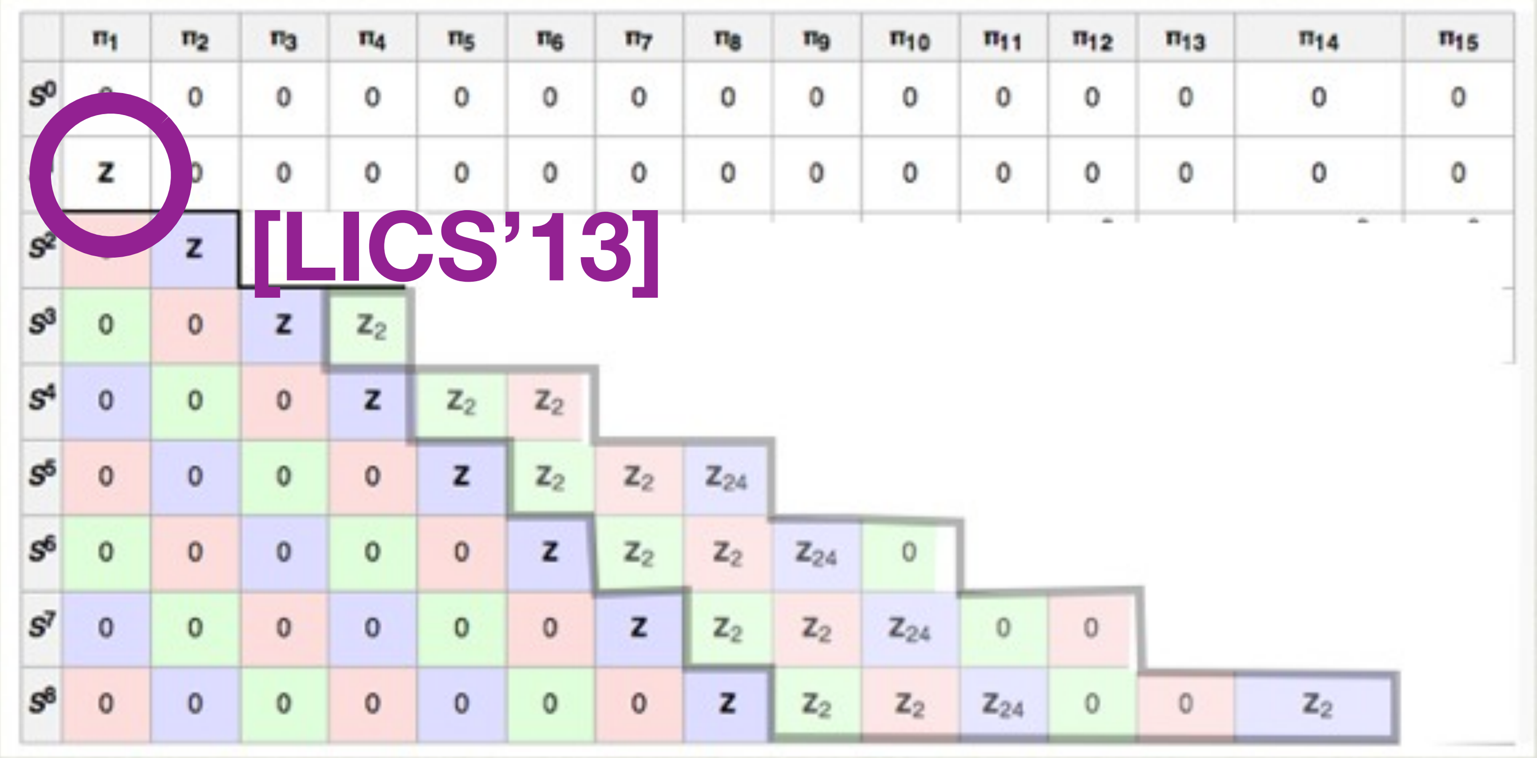
Computer-checked proofs

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)



	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	[illegible]	Z													
S^3	0	0	Z	Z_2											
S^4	0	0	0	Z	Z_2	Z_2									
S^5	0	0	0	0	Z	Z_2	Z_2	Z_{24}							
S^6	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0					
S^7	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0			
S^8	0	0	0	0	0	0	0	Z	Z_2	Z_2	Z_{24}	0	0	Z_2	

Fundamental group of circle

[LICS'13]

Two functions:

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

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

[LICS'13]

Two functions:

1. winding : (base = base) → $\mathbb{Z}$ **uses circle recursion**
2. $\text{loop}^n : \mathbb{Z} \to (\text{base} = \text{base})$

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

[LICS'13]

Two functions:

1. winding : (base = base) → $\mathbb{Z}$ **uses circle recursion**

2. loop^{n} : $\mathbb{Z}$ → (base = base)

Three proofs:

1. ∀n:$\mathbb{Z}$. winding(loop^{n}) = n

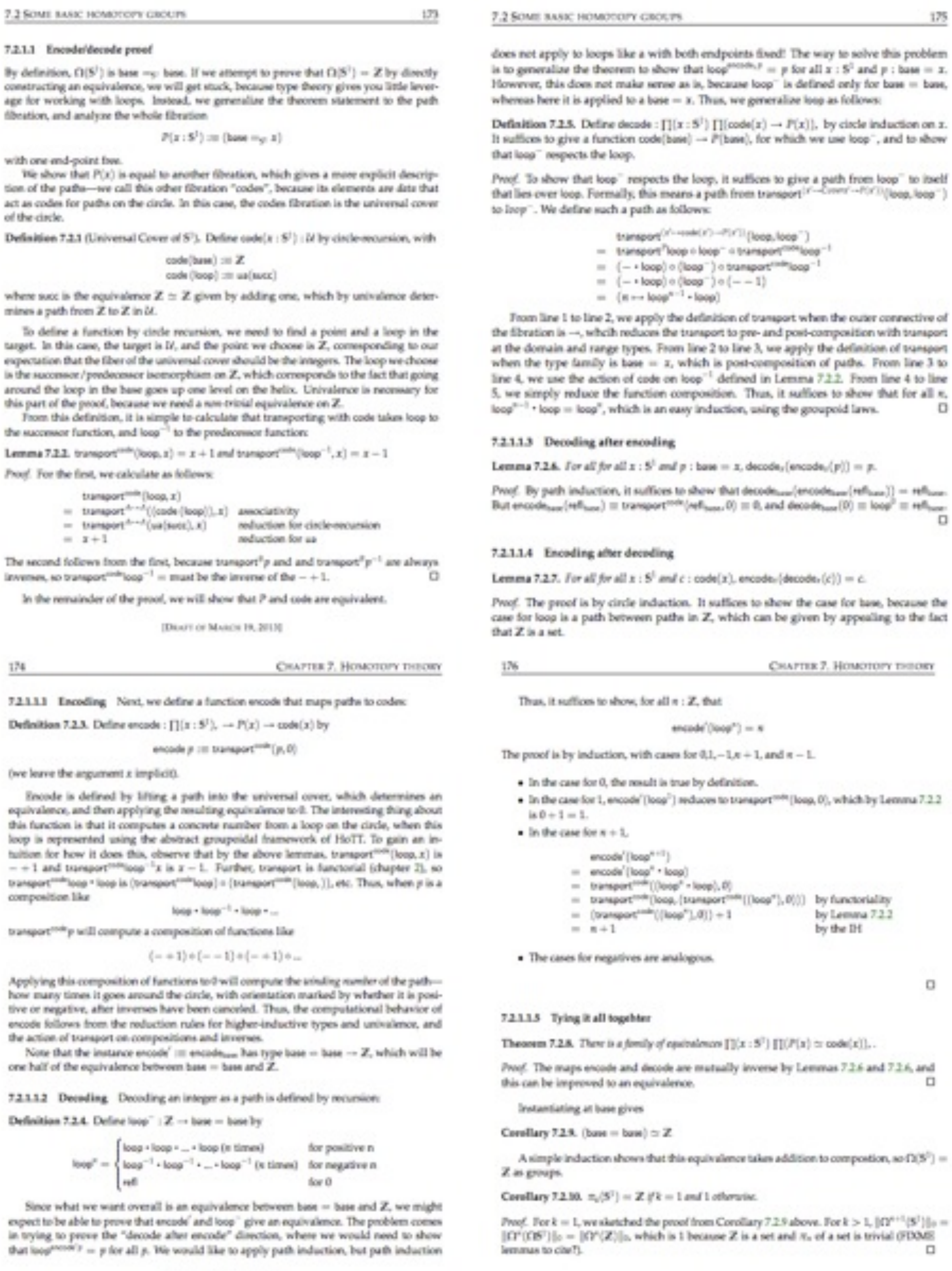
2. ∀p. $\text{loop}^{\text{winding(p)}}$ = p

3. ∀n,m. loop^{n+m} = loop^{n} o 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

```
conv1 : (Nat × String × ((Nat × Nat) × Nat))
      → (Nat × String × ((Nat × Nat) × 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 (_::_ (g (f x))) (map-fusion g f xs)

map-idfunc : ∀ {A} (l : List A) → map (\ x -> x) l ≡ l
map-idfunc [] = id
map-idfunc (x :: xs) = ap (_::_ x) (map-idfunc xs)

convert-inv : convert o convert ≡ (λ x -> x)
convert-inv = map conv1 o map conv1
               ≡〈 ! (λ≡ (map-fusion conv1 conv1)) 〉
              map (conv1 o conv1)
               ≡〈 id 〉
              map (\ x -> x)
               ≡〈 λ≡ map-idfunc 〉
              (λ x → x) ∎

convert-bijection : Bijection DB DB
convert-bijection =
  (convert ,
   is-bijection convert
             (λ x -> (ap≡ convert-inv))
             (λ x -> (ap≡ convert-inv)))
```

Homotopy Type Theory

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

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

There : Type -> Type
There A = List (Nat × String × A × Nat)

convert : DB → DB
convert = cast There swap

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

[{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)}]

→ convert ←

[{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)}]

[{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)}]

→ convert ←

[{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
   `swap(x,y) = (y,x)`

[{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)}]

→ convert ←

[{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
   `swap(x,y) = (y,x)`
2. Prove that `swap` is a bijection (it's self-inverse)

[{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)}]

→ convert ←

[{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
   `swap(x,y) = (y,x)`
2. Prove that `swap` is a bijection (it’s self-inverse)
3. Define a *parametrized type* describing where to swap:
   `There(X)=List{key:int, n:string, d:X×int}`

[{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)}]

convert

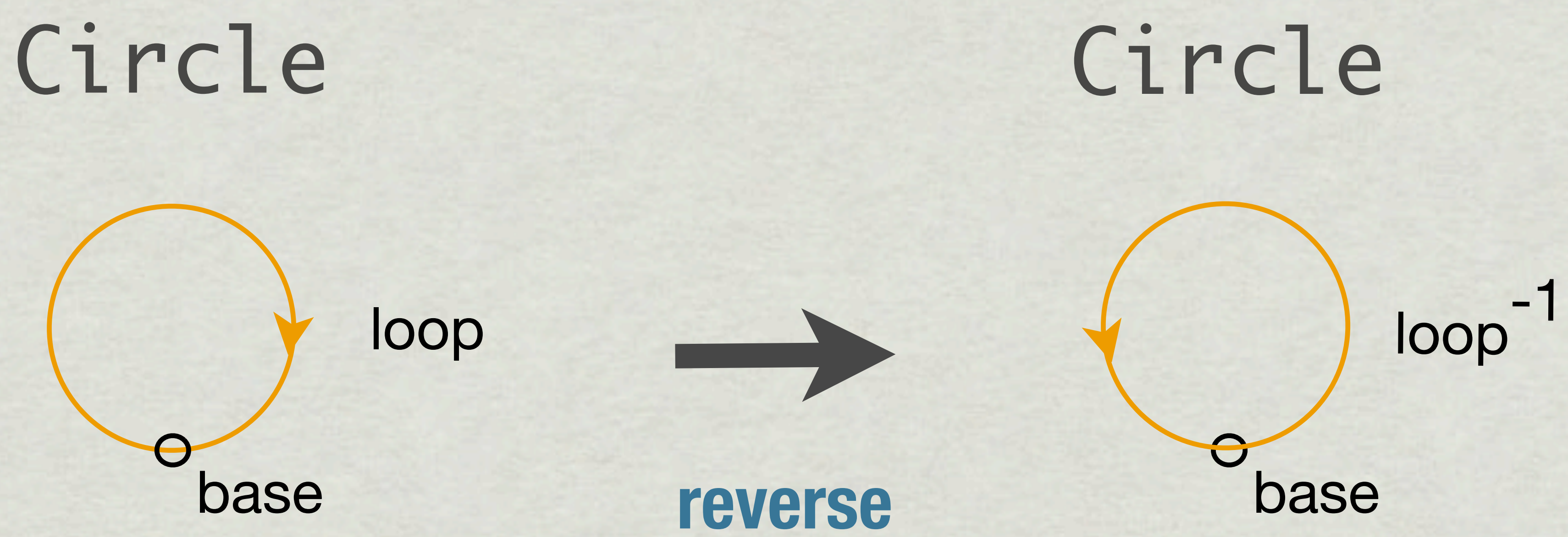
[{key=4,n="John",d=(5,30,1956)},
{key=8,n="Hugo",d=(12,29,1978)},
{key=15,n="James",d=(7,1,1968)},
{key=16,n="Sayid",d=(10,2,1967)},
{key=23,n="Jack",d=(12,3,1969)},
{key=42,n="Sun",d=(3,20,1980)}]

1. Define a function
   `swap(x,y) = (y,x)`
2. Prove that `swap` is a bijection (it's self-inverse)
3. Define a *parametrized type* describing where to swap:
   `There(X)=List{key:int, n:string, d:X×int}`
4. Define
   convert(db) = $\mathtt{cast}_{\mathtt{There}}$(swap,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

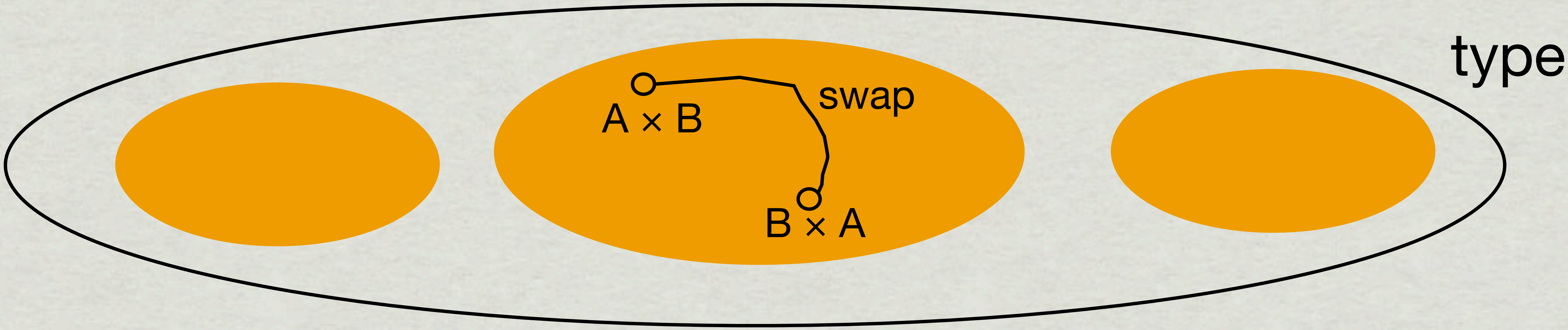
Functions on types

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

Functions on types

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

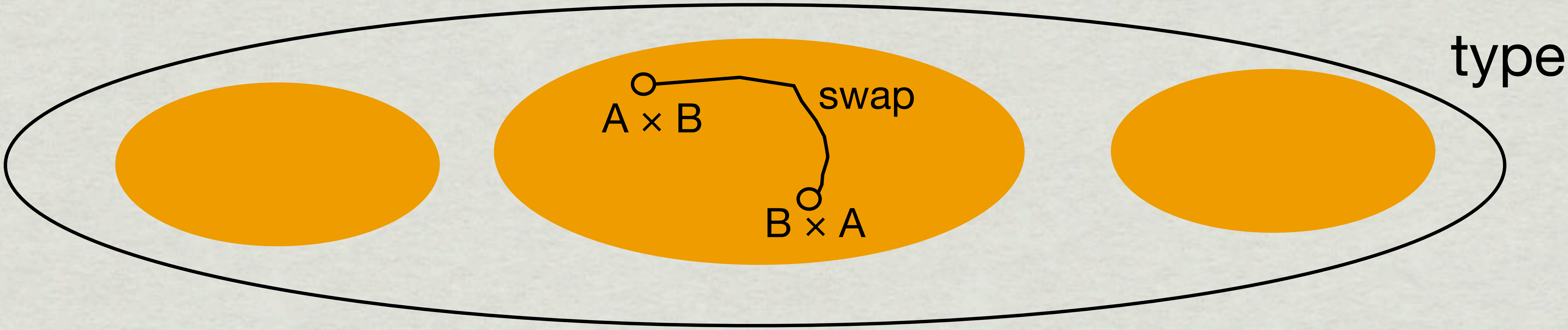
2. We define the paths between types to be **bijections**



Functions on types

1. `There(X)=List{key:int, n:string, d: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**



3. ∴ `There` gives an image for `swap`

Cast

$\texttt{cast}_{\texttt{There}}(\texttt{swap})$

applies `There` to `swap`: a *type-directed program* that builds bigger bijections from smaller ones

Computational interpretation of cast:

```
There(X)=List{key:int, n:string, d:X×int}
```

$\mathtt{cast}_{\mathtt{There}}\mathtt{(swap,db)}$

[{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:

```
There(X)=List{key:int, n:string, d:X×int}

   cast_There(swap,db)
=  map (cast_There1 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:

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

   cast_There(swap,db)
=  map (cast_There1 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:

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

   castThere(swap,db)
=  map (castThere1 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)}]

→ ←

[{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:

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

   cast_There(swap,db)
=  map (cast_There1 swap) db
=  map ({key,n,(d,m,y)} =>
         {key,n,(cast_Here(swap,(d,m)),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)}]

→ ←

[{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:

```
Here(X)=X

    cast_There(swap,db)
=   map (cast_There1 swap) db
=   map ({key,n,(d,m,y)} =>
            {key,n,(cast_Here(swap,(d,m)),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)}]

→ / ←

[{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:

```
Here(X)=X

   cast_There(swap,db)
=  map (cast_There1 swap) db
=  map ({key,n,(d,m,y)} =>
         {key,n,(cast_Here(swap,(d,m)),y)}) db
=  map ({key,n,(d,m,y)} =>
         {key,n,(m,d,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)}]

[{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)}]

Type theory

```
conv1 : (Nat × String × ((Nat × Nat) × Nat))
      → (Nat × String × ((Nat × Nat) × 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 (_::_ (g (f x))) (map-fusion g f xs)

map-idfunc : ∀ {A} (l : List A) → map (\ x -> x) l ≡ l
map-idfunc [] = id
map-idfunc (x :: xs) = ap (_::_ x) (map-idfunc xs)

convert-inv : convert o convert ≡ (λ x -> x)
convert-inv = map conv1 o map conv1
               ≡⟨ ! (λ≡ (map-fusion conv1 conv1)) ⟩
              map (conv1 o conv1)
               ≡⟨ id ⟩
              map (\ x -> x)
               ≡⟨ λ≡ map-idfunc ⟩
              (λ x → x) ∎

convert-bijection : Bijection DB DB
convert-bijection =
  (convert ,
   is-bijection convert
              (λ x -> (ap≡ convert-inv))
              (λ x -> (ap≡ convert-inv)))
```

Homotopy Type Theory

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

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

```
There : Type -> Type
There A = List (Nat × String × A × 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!

Identity and composition for $\Gamma \vdash \theta : \Delta$

$$\frac{\Gamma \supseteq \Delta}{\Gamma \vdash \mathsf{id}^{\Gamma}_{\Delta} : \Delta} \quad \frac{\Gamma_2 \vdash \theta_2 : \Gamma_3 \quad \Gamma_1 \vdash \theta_1 : \Gamma_2}{\Gamma_1 \vdash \theta_2[\theta_1] : \Gamma_3} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma_0 \vdash \delta : \theta_1 \simeq_{\Gamma} \theta_2}{\Gamma_0 \vdash \theta[\delta] : \theta[\theta_1] \simeq_{\Delta} \theta[\theta_2]}$$

$$\begin{array}{lcll}
\theta_0[\theta[\theta']] & \equiv & \theta_0[\theta][\theta'] & \textit{1-subst assoc/unit} \\
\theta_0[\mathsf{id}_{\Gamma}] & \equiv & \theta_0 & \\
\mathsf{id}[\theta] & \equiv & \theta & \\
\theta[\delta[\delta']] & \equiv & \theta[\delta][\delta'] & \textit{1-resp assoc} \\
\theta[\mathsf{refl}_{\theta'}] & \equiv & \mathsf{refl}_{\theta[\theta']} & \textit{1-resp preserves refl} \\
\theta[\theta'][\delta] & \equiv & \theta[\theta'[\delta]] & \textit{1-resp for 1-subst}
\end{array}$$

Identity, Inverses, and Composition for $\Gamma \vdash \delta : \theta \simeq_{\Delta} \theta'$

$$\frac{}{\Gamma \vdash \mathsf{refl}^{\Delta}_{\theta} : \theta \simeq_{\Delta} \theta} \quad \frac{\Gamma \vdash \delta : \theta_1 \simeq_{\Delta} \theta_2}{\Gamma \vdash \delta^{-1} : \theta_2 \simeq_{\Delta} \theta_1} \quad \frac{\Gamma \vdash \delta_1 : \theta_1 \simeq_{\Delta} \theta_2 \quad \Gamma \vdash \delta_2 : \theta_2 \simeq_{\Delta} \theta_3}{\Gamma \vdash \delta_2 \circ \delta_1 : \theta_1 \simeq_{\Delta} \theta_3} \quad \frac{\Gamma \vdash \delta : \theta \simeq_{\Delta} \theta' \quad \Gamma_0 \vdash \delta_0 : \theta_0 \simeq_{\Gamma} \theta'_0}{\Gamma_0 \vdash \delta[\delta_0] : \theta[\theta_0] \simeq_{\Delta} \theta'[\theta'_0]}$$

$$\begin{array}{lcll}
(\delta_3 \circ \delta_2) \circ \delta_1 & \equiv & \delta_3 \circ (\delta_2 \circ \delta_1) & \textit{trans assoc/unit} \\
(\delta \circ \mathsf{refl}) & \equiv & \delta & \\
(\mathsf{refl} \circ \delta) & \equiv & \delta & \\
(\delta \circ \delta^{-1}) & \equiv & \mathsf{refl} & \textit{inverse} \\
(\delta^{-1} \circ \delta) & \equiv & \mathsf{refl} &
\end{array}
\qquad
\begin{array}{lcll}
\delta_0[\delta[\delta']] & \equiv & \delta_0[\delta][\delta'] & \textit{2-resp assoc/unit} \\
\delta_0[\mathsf{refl}_{\mathsf{id}}] & \equiv & \delta_0 & \\
\mathsf{refl}_{\mathsf{id}_{\Gamma}}[\delta] & \equiv & \delta & \\
(\delta_1 \circ \delta_2)[\delta_3 \circ \delta_4] & \equiv & \delta_1[\delta_3] \circ \delta_2[\delta_4] & \textit{interchange} \\
\mathsf{refl}_{\theta}[\delta] & \equiv & \theta[\delta] & \textit{delegate}
\end{array}$$

Composition for $\Gamma \vdash A$ type

$$\frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash A \ \mathsf{type}}{\Gamma \vdash A[\theta] \ \mathsf{type}} \quad \frac{\Delta\ \mathsf{ctx} \quad \Delta \vdash C \ \mathsf{type} \quad \Gamma \vdash \delta : \theta_1 \simeq_{\Delta} \theta_2 \quad \Gamma \vdash M : C[\theta_1]}{\Gamma \vdash \mathsf{map}_{\Delta.C}\ \delta\ M : C[\theta_2]}$$

$$\begin{array}{rcll}
A[\theta[\theta']] & \equiv & A[\theta][\theta'] & \textit{0-subst assoc/unit} \\
A[\mathsf{id}_{\Gamma}] & \equiv & A & \\
\mathsf{map}_{\Delta.C}\ \mathsf{refl}_{\theta}\ M & \equiv & M & \textit{0-resp functoriality} \\
\mathsf{map}_{\Delta.C}\ (\delta_2 \circ \delta_1)\ M & \equiv & \mathsf{map}_{\Delta.C}\ \delta_2\ (\mathsf{map}_{\Delta.C}\ \delta_1\ M) & \\
(\mathsf{map}_{\Delta.C}\ \delta\ M)[\theta_0] & \equiv & \mathsf{map}_{\Delta.C}\ \delta[\mathsf{refl}_{\theta_0}]\ M[\theta_0] & \textit{1-subst for map} \\
(\mathsf{map}_{C}\ (\delta : \theta_1 \simeq \theta_2)\ M)[\delta' : \theta'_1 \simeq \theta'_2] & \equiv & \mathsf{resp}\ (x.\mathsf{map}_{C}\ (\delta[\mathsf{refl}_{\theta'_2}])\ x)\ (M[\delta']) & \textit{1-resp for map} \\
\mathsf{map}_{\Delta.C[\theta : \Delta']}\ \delta\ M & \equiv & \mathsf{map}_{\Delta'.C}\ \mathsf{refl}_{\theta}[\delta]\ M & \textit{def. map for } A[\theta]
\end{array}$$

Composition for $\Gamma \vdash M : A$

$$\frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash M : A}{\Gamma \vdash M[\theta] : A[\theta]} \quad \frac{\Delta \vdash M : A \quad \Gamma \vdash \delta : \theta_1 \simeq_{\Delta} \theta_2}{\Gamma \vdash M[\delta] : (\mathsf{map}_{\Delta.A}\ \delta\ (M[\theta_1])) \simeq_{A[\theta_2]} M[\theta_2]}$$

$$\begin{array}{lcll}
M[\theta[\theta']] & \equiv & M[\theta][\theta'] & \textit{1-subst assoc/unit} \\
M[\mathsf{id}_{\Gamma}] & \equiv & M & \\
M[\delta[\delta']] & \equiv & M[\delta][\delta'] & \textit{1-resp assoc/unit} \\
M[\mathsf{refl}_{\theta}] & \equiv & \mathsf{refl}_{M[\theta]} & \textit{1-resp preserves refl} \\
M[\theta][\delta] & \equiv & M[\theta[\delta]] & \textit{1-resp for 1-subst}
\end{array}$$

Identity, Inverses, and Composition for $\Gamma \vdash \alpha : M \simeq_{A} N$

$$\frac{}{\Gamma \vdash \mathsf{refl}^{A}_{M} : M \simeq_{A} M} \quad \frac{\Gamma \vdash \alpha : M_1 \simeq_{A} M_2}{\Gamma \vdash \alpha^{-1} : M_2 \simeq_{A} M_1} \quad \frac{\Gamma \vdash \alpha_1 : M_1 \simeq_{A} M_2 \quad \Gamma \vdash \alpha_2 : M_2 \simeq_{A} M_3}{\Gamma \vdash \alpha_2 \circ \alpha_1 : M_1 \simeq_{A} M_3} \quad \frac{\Gamma_0 \vdash \delta_0 : \theta_0 \simeq_{\Gamma} \theta'_0 \quad \Gamma \vdash \alpha : M \simeq_{A} N}{\Gamma_0 \vdash \alpha[\delta_0] : (\mathsf{map}_{\Gamma.A}\ \delta_0\ (M[\theta_0])) \simeq_{A[\theta'_0]} N[\theta'_0]}$$

$$\begin{array}{lcll}
(\alpha_3 \circ \alpha_2) \circ \alpha_1 & \equiv & \alpha_3 \circ (\alpha_2 \circ \alpha_1) & \textit{trans assoc/unit} \\
(\alpha \circ \mathsf{refl}) & \equiv & \alpha & \\
(\mathsf{refl} \circ \alpha) & \equiv & \alpha & \\
(\alpha \circ \alpha^{-1}) & \equiv & \mathsf{refl} & \textit{inverse} \\
(\alpha^{-1} \circ \alpha) & \equiv & \mathsf{refl} & \\
\alpha[\delta[\delta']] & \equiv & \alpha[\delta][\delta'] & \textit{2-resp assoc/unit} \\
\alpha[\mathsf{refl}_{\mathsf{id}}] & \equiv & \alpha & \\
(\alpha_1 \circ \alpha_2)[\delta_3 \circ \delta_4] & \equiv & \alpha_1[\delta_3] \circ \mathsf{resp}\ (x.\mathsf{map}\ \delta_3\ x)\ (\alpha_2[\delta_4]) & \textit{interchange} \\
\mathsf{refl}_{M}[\delta] & \equiv & M[\delta] & \textit{delegate}
\end{array}$$

Omitted Rules: All judgements respect equality; all equality judgements are congruences.

Derived forms:

$$\begin{array}{lll}
\mathsf{resp}\ (x.F)\ \alpha & \text{means} & (x.F)[\mathsf{refl}_{\mathsf{id}}, \alpha/x] \\
\mathsf{map}^{1}_{x:A.B}\ \alpha\ M & \text{means} & \mathsf{map}_{\Gamma,x:A.B}\ (\mathsf{refl}_{\mathsf{id}}, \alpha/x)\ M
\end{array}$$

Figure 1. Judgemental Presentation of Equivalence

$$\frac{}{\Gamma \vdash \mathsf{set}\ \mathsf{type}} \quad \frac{\Gamma \vdash S : \mathsf{set}}{\Gamma \vdash El(S)\ \mathsf{type}}$$

$$\frac{\Gamma \vdash \alpha : M \simeq_{El(S)} N}{\Gamma \vdash M \equiv N} \quad \frac{\Gamma \vdash \alpha : M \simeq_{El(S)} N}{\Gamma \vdash \alpha \equiv \mathsf{refl}_{M}}$$

$$\begin{array}{rcll}
\mathsf{set}[\theta_0] & \equiv & \mathsf{set} & \textit{0-subst} \\
(El(S))[\theta_0] & \equiv & El(S[\theta_0]) & \\
\mathsf{map}_{\Delta.\mathsf{set}}\ \delta\ M & \equiv & M & \textit{0-resp} \\
-^{-1}, - \circ - \text{ for set} & & \text{groupoid laws plus below} & \\
\mathsf{refl}_{M : El(S)} & & \text{is canonical} & \\
-^{-1}, - \circ -, \text{2-resp for } El(S) & & \text{trivial by uip} &
\end{array}$$

$$\frac{}{\Gamma \vdash \mathsf{bool} : \mathsf{set}} \ (\mathsf{refl}_{\mathsf{bool}} \text{ is canonical}) \quad \frac{}{\Gamma \vdash \mathsf{not} : \mathsf{bool} \simeq_{\mathsf{set}} \mathsf{bool}} \quad \frac{\Gamma \vdash M : 2}{\Gamma \vdash \mathsf{in}\ M : El(\mathsf{bool})} \quad \frac{\Gamma \vdash M : El(\mathsf{bool})}{\Gamma \vdash \mathsf{out}\ M : 2}$$

$$\begin{array}{rclll}
\mathsf{out}\ (\mathsf{in}\ M) & \equiv & M & & \textit{1-}\beta \\
\mathsf{in}\ (\mathsf{out}\ M) & \equiv & M & & \textit{1-}\eta \\
\mathsf{map}_{\Delta.El(S)}\ \delta\ \mathsf{true} & \equiv & \mathsf{false} & \text{if } S[\delta] \equiv \mathsf{not} & \textit{0-resp} \\
\mathsf{map}_{\Delta.El(S)}\ \delta\ \mathsf{false} & \equiv & \mathsf{true} & \text{if } S[\delta] \equiv \mathsf{not} & \textit{0-resp}
\end{array}
\qquad
\begin{array}{rcll}
\mathsf{bool}[\theta] & \equiv & \mathsf{bool} & \textit{1-subst} \\
(\mathsf{in}\ M)[\theta] & \equiv & \mathsf{in}\ M[\theta] & \textit{1-subst} \\
(\mathsf{out}\ M)[\theta] & \equiv & \mathsf{out}\ M[\theta] & \textit{1-subst} \\
\mathsf{bool}[\delta] & \equiv & \mathsf{refl}_{\mathsf{bool}} & \textit{1-resp} \\
\mathsf{not}^{-1} & \equiv & \mathsf{not} & \textit{sym} \\
\mathsf{not}[\delta] & \equiv & \mathsf{not} & \textit{2-resp}
\end{array}$$

Figure 5. Universe

Empty context:

$$\frac{}{\cdot\ \mathsf{ctx}} \quad (\mathsf{id}.\ \text{is the only canonical substitution}) \quad (\mathsf{refl}_{\mathsf{id}.}\ \text{is the only canonical equivalence})
\qquad
\begin{array}{lcll}
\theta : \cdot & \equiv & \mathsf{id}. & \textit{1-}\eta \\
\delta : \theta \simeq. \theta' & \equiv & \mathsf{refl}. & \textit{2-}\eta
\end{array}$$

Term variables:

$$\frac{\Gamma\ \mathsf{ctx} \quad \Gamma \vdash A\ \mathsf{type}}{\Gamma, x{:}A\ \mathsf{ctx}} \quad \frac{x{:}A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma \vdash M : A[\theta]}{\Gamma \vdash \theta, M/x : \Delta, x : A} \quad \frac{\Gamma \vdash \delta : \theta \simeq_{\Delta} \theta' \quad \Gamma \vdash \alpha : (\mathsf{map}_{\Delta.A}\ \delta\ M) \simeq_{A[\theta']} N}{\Gamma \vdash (\delta, \alpha/x) : (\theta, M/x) \simeq_{\Delta, x:A} (\theta', N/x)}$$

$$\begin{array}{rcll}
\mathsf{id}^{\Gamma,x:A}_{\Gamma}[\theta, M/x] & \equiv & \theta & \textit{1-}\beta \\
x[\theta, M/x] & \equiv & M & \textit{1-}\beta \\
\theta : (\Gamma, x{:}A) & \equiv & \mathsf{id}_{\Gamma}[\theta], x[\theta]/x & \textit{1-}\eta \\
\mathsf{id}^{\Gamma,x:A}_{\Gamma}[\delta, \alpha/x] & \equiv & \delta & \textit{2-}\beta \\
x[\delta, \alpha/x] & \equiv & \alpha & \textit{2-}\beta \\
\delta : \theta \simeq_{(\Gamma,x:A)} \theta' & \equiv & \mathsf{id}_{\Gamma}[\delta], x[\delta]/x & \textit{2-}\eta
\end{array}
\qquad
\begin{array}{rcll}
\mathsf{id}_{\Gamma,x:A} & \equiv & \mathsf{id}_{\Gamma}, x/x & \textit{1-id} \\
(\theta, M/x)[\theta_0] & \equiv & \theta[\theta_0], M[\theta_0]/x & \textit{1-subst} \\
(\theta, M/x)[\delta_0] & \equiv & \theta[\delta_0], M[\delta_0]/x & \textit{1-resp} \\
\mathsf{refl}_{\theta, M/x} & \equiv & \mathsf{refl}_{\theta}, \mathsf{refl}_{M}/x & \textit{refl} \\
(\delta, \alpha/x)^{-1} & \equiv & (\delta^{-1}, (\mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta^{-1}\ x)\ \alpha^{-1})/x) & \textit{sym} \\
(\delta_2, \alpha_2/x) \circ (\delta_1, \alpha_1/x) & \equiv & (\delta_2 \circ \delta_1), (\alpha_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta_2\ x)\ \alpha_1)/x & \textit{trans} \\
(\delta, \alpha/x)[\delta_0] & \equiv & \delta[\delta_0], \alpha[\delta_0]/x & \textit{2-resp}
\end{array}$$

Figure 2. Contexts

$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma, x{:}A \vdash B\ \mathsf{type}}{\Gamma \vdash \Pi\, x{:}A.\, B\ \mathsf{type}} \quad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.\, M : \Pi\, x{:}A.\, B} \quad \frac{\Gamma \vdash M_1 : \Pi\, x{:}A.\, B \quad \Gamma \vdash M_2 : A}{\Gamma \vdash M_1\ M_2 : B[M_2/x]} \quad \frac{\Gamma, x{:}A \vdash \alpha : (M\ x) \simeq_{B} (N\ x)}{\Gamma \vdash \lambda x.\, \alpha : M \simeq_{\Pi\, x:A.\, B} N} \quad \frac{\Gamma \vdash \alpha : M \simeq_{\Pi\, x:A.\, B} N \quad \Gamma \vdash \beta : M_1 \simeq_{A} N_1}{\Gamma \vdash \alpha\ \beta : \mathsf{map}^{1}_{B}\ \beta\ (M\, M_1) \simeq_{B[N_1/x]} (N N_1)}$$

$$\begin{array}{lcll}
(\lambda x.\, M)\ N & \equiv & M[N/x] & \textit{1-}\beta \\
M : \Pi\, x{:}A.\, B & \equiv & \lambda x.\, M\ x & \textit{1-}\eta \\
(\lambda x.\, \alpha_1)\ \alpha_2 & \equiv & \alpha_1[\mathsf{refl}, \alpha_2/x] & \textit{2-}\beta \\
\alpha : M \simeq_{\Pi\, x:A.\, B} N & \equiv & \lambda x.\, \alpha\ (\mathsf{refl}_x) & \textit{2-}\eta \\
(\Pi\, x{:}A.\, B)[\theta_0] & \equiv & \Pi\, x{:}A[\theta_0].\, B[\theta_0, x/x] & \textit{0-subst} \\
\mathsf{map}_{\Delta.\Pi\, x:A.\, B}\ \delta\ M & \equiv & & \textit{0-resp} \\
\multicolumn{4}{l}{\quad \lambda x.\, \mathsf{map}_{\Delta, x:A.B}\ (\delta, \mathsf{refl}/x)\ (M\ (\mathsf{map}_{\Delta.A}\ \delta^{-1}\ x))}
\end{array}
\qquad
\begin{array}{lcll}
(\lambda x.\, M)[\theta_0] & \equiv & \lambda x.\, M[(\theta_0, x/x)] & \textit{1-subst} \\
(M_1\ M_2)[\theta_0] & \equiv & (M_1[\theta_0])\ (M_2[\theta_0]) & \textit{1-subst} \\
(\lambda x.\, M)[\delta] & \equiv & \lambda x.\, M[\delta, \mathsf{refl}/x] & \textit{1-resp} \\
(M\ N)[\delta] & \equiv & M[\delta]\ N[\delta] & \textit{1-resp} \\
\mathsf{refl}_{M} & \equiv & \lambda x.\, \mathsf{refl}_{M\ x} & \textit{refl} \\
(\lambda x.\, \alpha)^{-1} & \equiv & \lambda x.\, \alpha^{-1} & \textit{sym} \\
(\lambda x.\, \alpha_2) \circ (\lambda x.\, \alpha_1) & \equiv & \lambda x.\, \alpha_2 \circ \alpha_1 & \textit{trans} \\
(\lambda x{:}A.\, \alpha)[\delta_0] & \equiv & \lambda x{:}A[\theta'_0].\, \alpha[\delta_0, \mathsf{refl}/x] & \textit{2-resp} \\
(\alpha_1\ \alpha_2)[\delta_0] & \equiv & (\alpha_1[\delta_0])\ (\alpha_2[\delta_0]) & \textit{2-resp}
\end{array}$$

Figure 3. Π-Types

$$\frac{}{\Gamma \vdash 2\ \mathsf{type}} \quad \frac{}{\Gamma \vdash \mathsf{true} : 2} \quad \frac{}{\Gamma \vdash \mathsf{false} : 2} \quad \frac{\Gamma \vdash M : 2 \quad \Gamma, x{:}2 \vdash C\ \mathsf{type} \quad \Gamma \vdash M_1 : C[\mathsf{true}/x] \quad \Gamma \vdash M_2 : C[\mathsf{false}/x]}{\Gamma \vdash \mathsf{if}_{x.C}(M, M_1, M_2) : C[M/x]}$$

$$\frac{\Gamma \vdash M : 2 \quad \Gamma, x{:}2 \vdash C\ \mathsf{type} \quad \Gamma, x{:}2 \vdash M_1, M_2 : C \quad \Gamma \vdash \alpha_1 : M_1[\mathsf{true}/x] \simeq_{C[\mathsf{true}/x]} M_2[\mathsf{true}/x] \quad \Gamma \vdash \alpha_2 : M_1[\mathsf{false}/x] \simeq_{C[\mathsf{false}/x]} M_2[\mathsf{false}/x]}{\Gamma \vdash \mathsf{if}_{x.C}(M, \alpha_1, \alpha_2) : M_1[M/x] \simeq_{C[M/x]} M_2[M/x]}$$

$$(\mathsf{refl}_{\mathsf{true}} \text{ and } \mathsf{refl}_{\mathsf{false}} \text{ are canonical}) \quad \frac{\Gamma \vdash \alpha : M \simeq_2 N}{\Gamma \vdash M \equiv N}\ \textit{reflection} \quad \frac{\Gamma \vdash \alpha : M \simeq_2 N}{\Gamma \vdash \alpha \equiv \mathsf{refl}_{M}}\ \textit{uip} \quad \frac{\Gamma \vdash \alpha : \mathsf{true} \simeq_2 \mathsf{false}}{\Gamma \vdash \mathcal{J}} \ (\text{where } \mathcal{J} \text{ is any judgement})$$

$$\begin{array}{rcll}
\mathsf{if}(\mathsf{true}, M_1, M_2) & \equiv & M_1 & \textit{1-}\beta \\
\mathsf{if}(\mathsf{false}, M_1, M_2) & \equiv & M_2 & \textit{1-}\beta \\
M[(N : El(2))/x] & \equiv & \mathsf{if}(N, M[\mathsf{true}/x], M[\mathsf{false}/x]) & \textit{1-}\eta \\
\mathsf{if}_{x.C}(\mathsf{true}, \alpha_1, \alpha_2) & \equiv & \alpha_1 & \textit{12-}\beta \\
\mathsf{if}_{x.C}(\mathsf{false}, \alpha_1, \alpha_2) & \equiv & \alpha_2 & \textit{12-}\beta \\
\alpha[(\mathsf{refl}_{M:2})/x] & \equiv & \mathsf{if}(M, \alpha[\mathsf{refl}_{\mathsf{true}}/x], \alpha[\mathsf{refl}_{\mathsf{false}}/x]) & \textit{12-}\eta \\
2[\theta] & \equiv & 2 & \textit{0-subst} \\
\mathsf{map}_{\Delta.2}\ \delta\ M & \equiv & M & \textit{0-resp}
\end{array}$$

$$\begin{array}{rcll}
\mathsf{true}[\theta] & \equiv & \mathsf{true} & \textit{1-subst} \\
\mathsf{false}[\theta] & \equiv & \mathsf{false} & \textit{1-subst} \\
\mathsf{if}_{x:2.C}(M, M_1, M_2)[\theta] & \equiv & & \textit{1-subst} \\
\multicolumn{4}{l}{\quad \mathsf{if}_{x:2.C[\theta, x/x]}(M[\theta], M_1[\theta], M_2[\theta])} \\
\mathsf{true}[\delta] & \equiv & \mathsf{refl}_{\mathsf{true}} & \textit{1-resp} \\
\mathsf{false}[\delta] & \equiv & \mathsf{refl}_{\mathsf{false}} & \textit{1-resp} \\
\mathsf{if}_{x:2.C}(M, M_1, M_2)[\delta : \theta_1 \simeq \theta_2] & \equiv & & \textit{1-resp} \\
\multicolumn{4}{l}{\quad \mathsf{if}_{x:2.C[\theta_2, x/x]}(M[\theta_2], M_1[\delta], M_2[\delta])} \\
\mathsf{refl} & & \text{is canonical} & \\
-^{-1}, - \circ - & & \text{trivial by uip} & \\
\mathsf{if}(M, \alpha_1, \alpha_2)[\delta : \theta_1 \simeq \theta_2] & \equiv & \mathsf{if}(M[\theta_2], \alpha_1[\delta], \alpha_2[\delta]) & \textit{2-resp}
\end{array}$$

Figure 4. Booleans (as an Extensional Set)

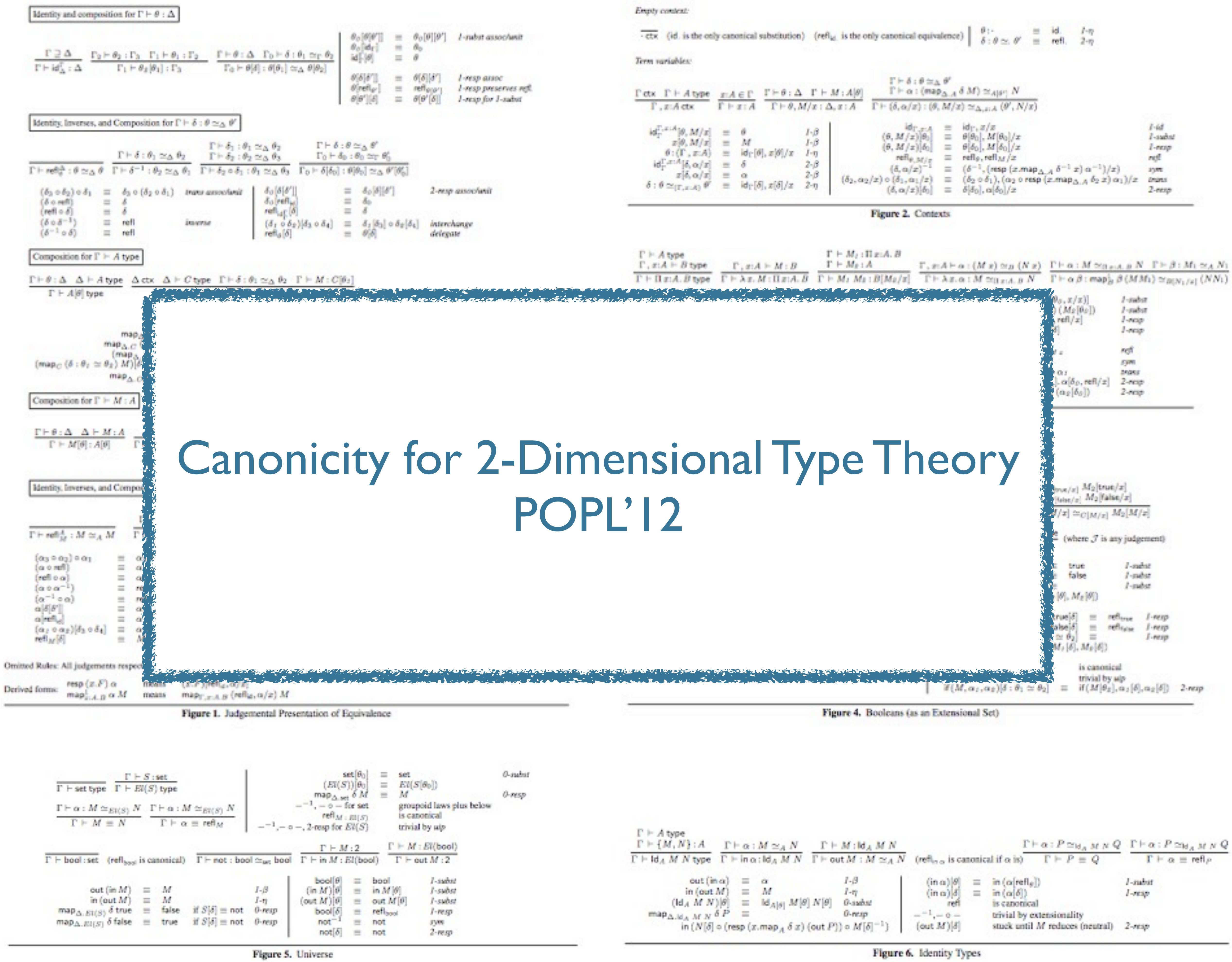
$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash \{M, N\} : A}{\Gamma \vdash \mathsf{Id}_A\ M\ N\ \mathsf{type}} \quad \frac{\Gamma \vdash \alpha : M \simeq_A N}{\Gamma \vdash \mathsf{in}\ \alpha : \mathsf{Id}_A\ M\ N} \quad \frac{\Gamma \vdash M : \mathsf{Id}_A\ M\ N}{\Gamma \vdash \mathsf{out}\ M : M \simeq_A N} \quad (\mathsf{refl}_{\mathsf{in}\ \alpha} \text{ is canonical if } \alpha \text{ is}) \quad \frac{\Gamma \vdash \alpha : P \simeq_{\mathsf{Id}_A\ M\ N} Q}{\Gamma \vdash P \equiv Q} \quad \frac{\Gamma \vdash \alpha : P \simeq_{\mathsf{Id}_A\ M\ N} Q}{\Gamma \vdash \alpha \equiv \mathsf{refl}_P}$$

$$\begin{array}{rcll}
\mathsf{out}\ (\mathsf{in}\ \alpha) & \equiv & \alpha & \textit{1-}\beta \\
\mathsf{in}\ (\mathsf{out}\ M) & \equiv & M & \textit{1-}\eta \\
(\mathsf{Id}_A\ M\ N)[\theta] & \equiv & \mathsf{Id}_{A[\theta]}\ M[\theta]\ N[\theta] & \textit{0-subst} \\
\mathsf{map}_{\Delta.\mathsf{Id}_A\ M\ N}\ \delta\ P & \equiv & & \textit{0-resp} \\
\multicolumn{4}{l}{\quad \mathsf{in}\ (N[\delta] \circ (\mathsf{resp}\ (x.\mathsf{map}_A\ \delta\ x)\ (\mathsf{out}\ P)) \circ M[\delta]^{-1})}
\end{array}
\qquad
\begin{array}{rcll}
(\mathsf{in}\ \alpha)[\theta] & \equiv & \mathsf{in}\ (\alpha[\mathsf{refl}_{\theta}]) & \textit{1-subst} \\
(\mathsf{in}\ \alpha)[\delta] & \equiv & \mathsf{in}\ (\alpha[\delta]) & \textit{1-resp} \\
\mathsf{refl} & & \text{is canonical} & \\
-^{-1}, - \circ - & & \text{trivial by extensionality} & \\
(\mathsf{out}\ M)[\delta] & & \text{stuck until } M \text{ reduces (neutral)} & \textit{2-resp}
\end{array}$$

Figure 6. Identity Types

Canonicity for 2-Dimensional Type Theory

POPL'12



More applications

- For modular code, can reason about a fast implementation using a reference implementation: `cast` a proof about the reference implementation to the fast implementation
- Can program domain-specific program verification logics, using `cast` to implement the *structural properties* [thesis + MFPS'11]

Conclusion

Types are ∞-groupoids

type theory

<program> : <type>

<proof> : <prog$_1$> = <prog$_2$>

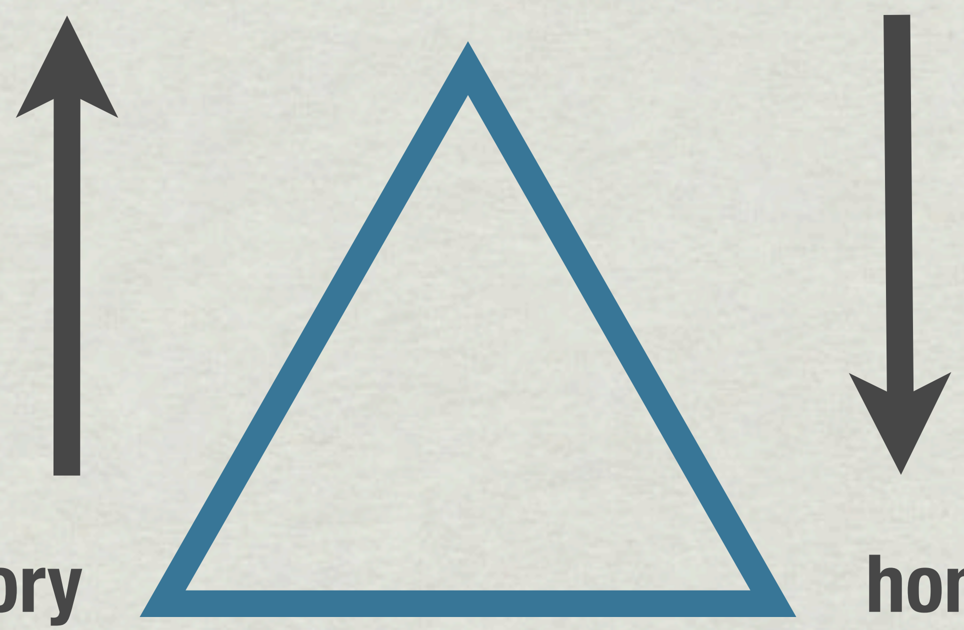
<2-proof> : <proof$_1$> = <proof$_2$>

<3-proof> : <2-proof$_1$> = <2-proof$_2$>

⋮

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

Homotopy Type Theory



Papers and code

1. Fundamental group of the circle [LICS'13]
   Formal 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

```
signature SEQUENCE =
sig
  type 'a seq

  val length : 'a seq -> int
  val nth    : int -> 'a seq -> 'a
  val tabulate : (int -> 'a) -> int -> 'a seq

  val map : ('a -> 'b) -> 'a seq -> 'b seq
  val reduce : (('a * 'a) -> 'a) -> 'a -> 'a seq -> 'a

end
```

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