Functional programs that prove theorems about spaces

Dan Licata

Wesleyan University
Department of Mathematics and Computer Science

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

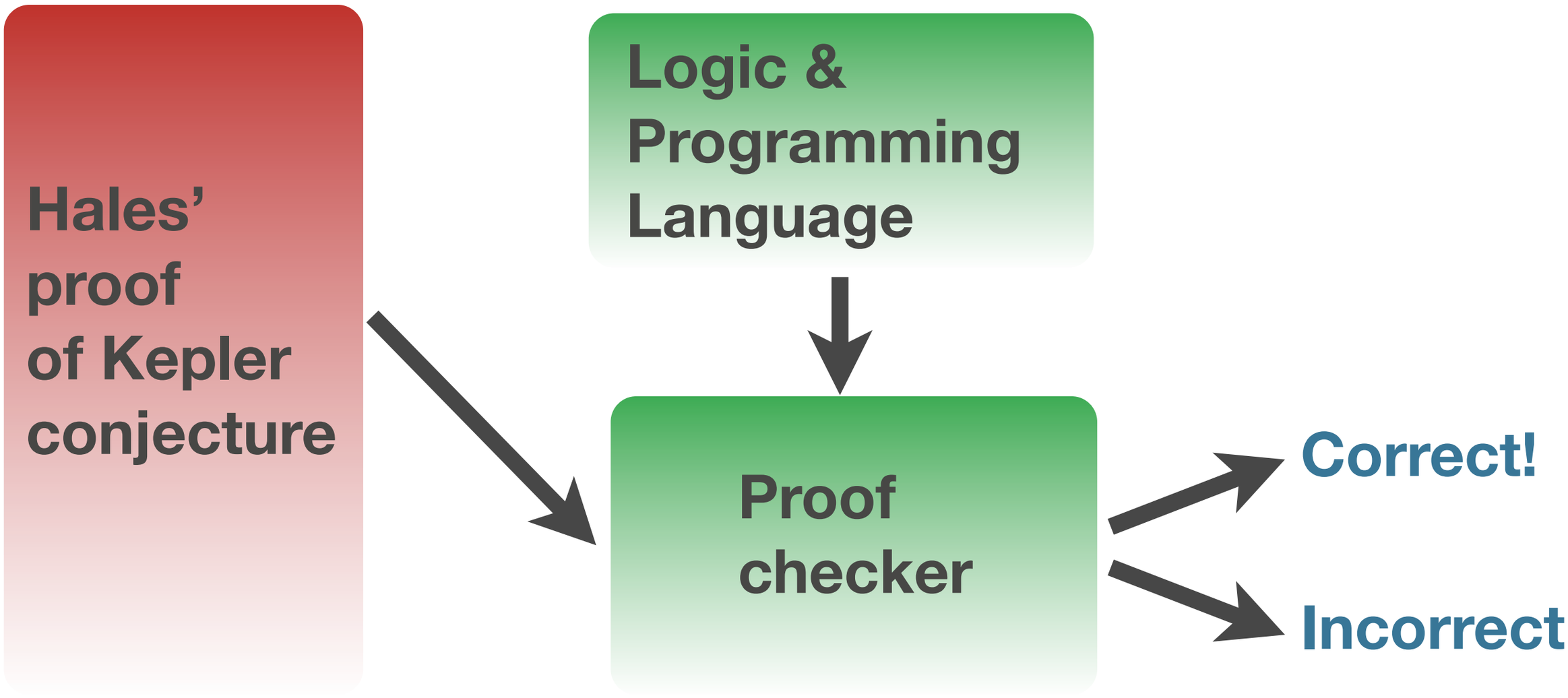
In 2003, after 4 years' work,
12 referees had checked lots of lemmas,
but gave up on verifying the proof

Proofs can be hard to check

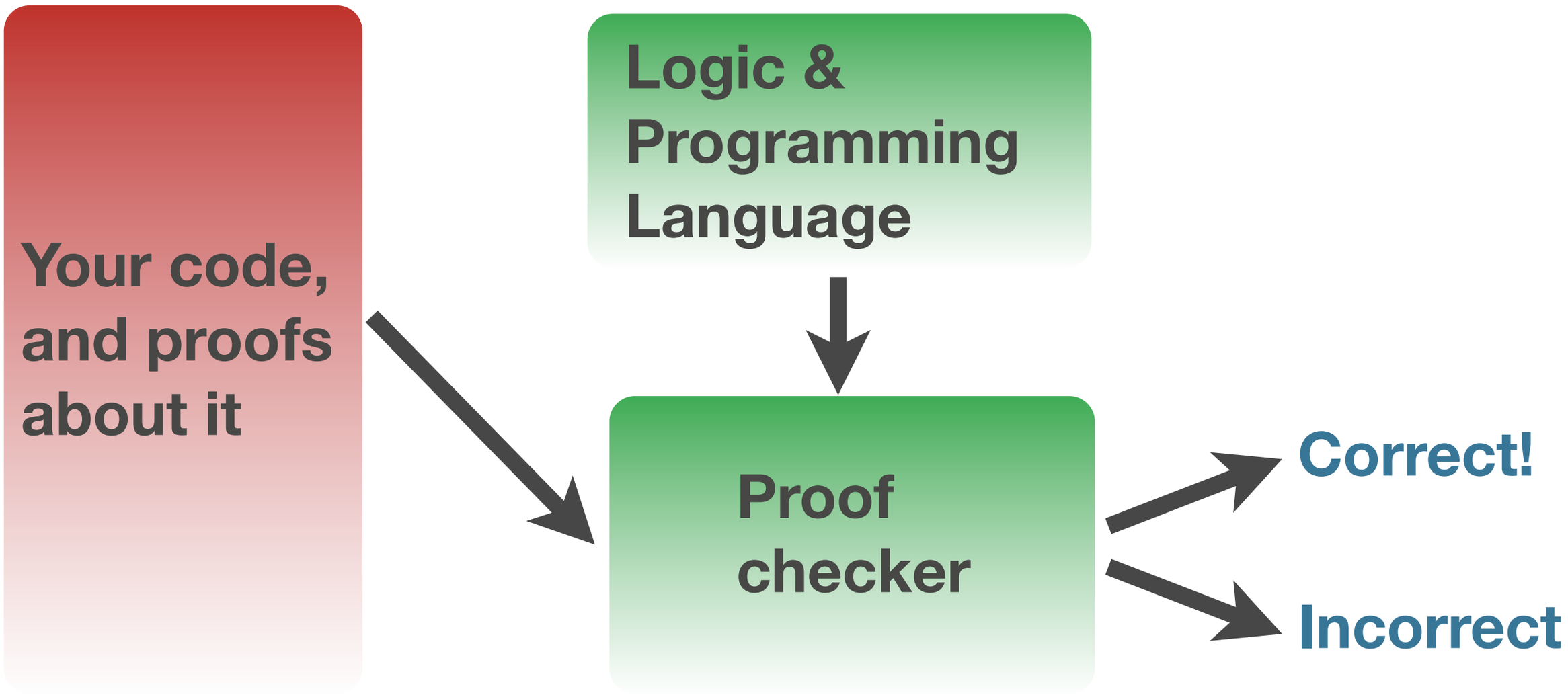
In 2003, after 4 years' work,
12 referees had checked lots of lemmas,
but gave up on verifying the proof

"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



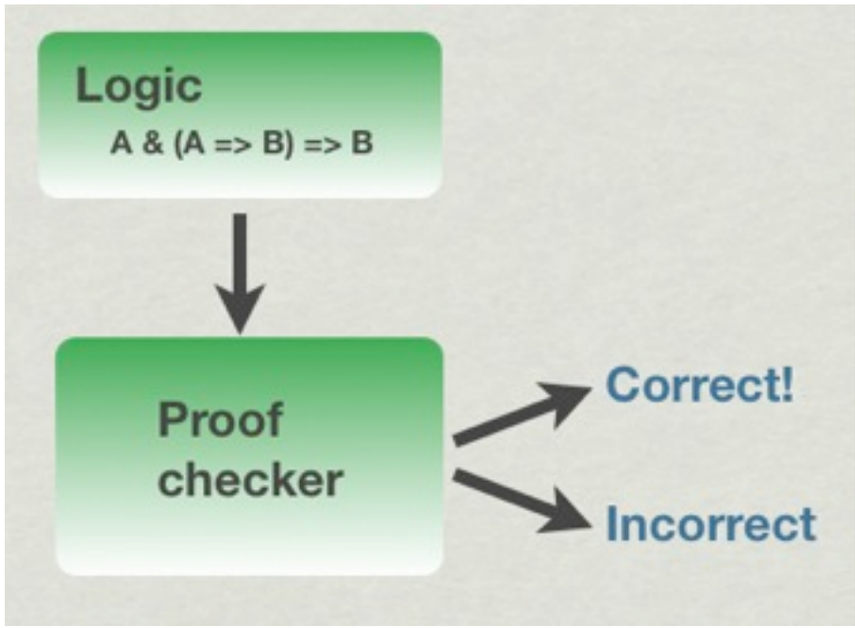
Computer-checked software



Computer-assisted proofs

Proof assistant

- Interactive proof editor
- Automated proofs
- Libraries



Kepler proof

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

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

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

Informal

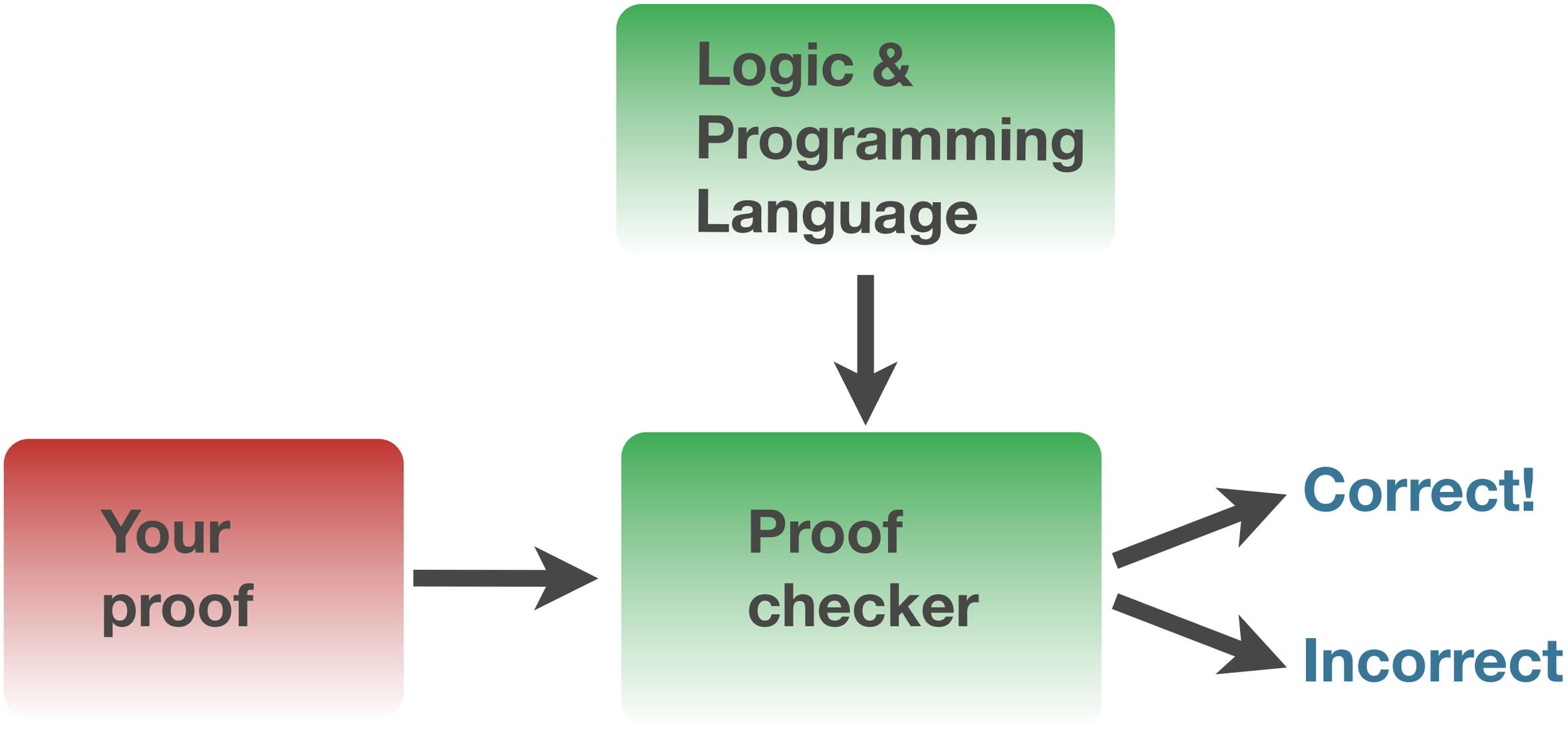
- 300 pages of math + 15,000 lines of code
- 15 hours to run

Computer-checked

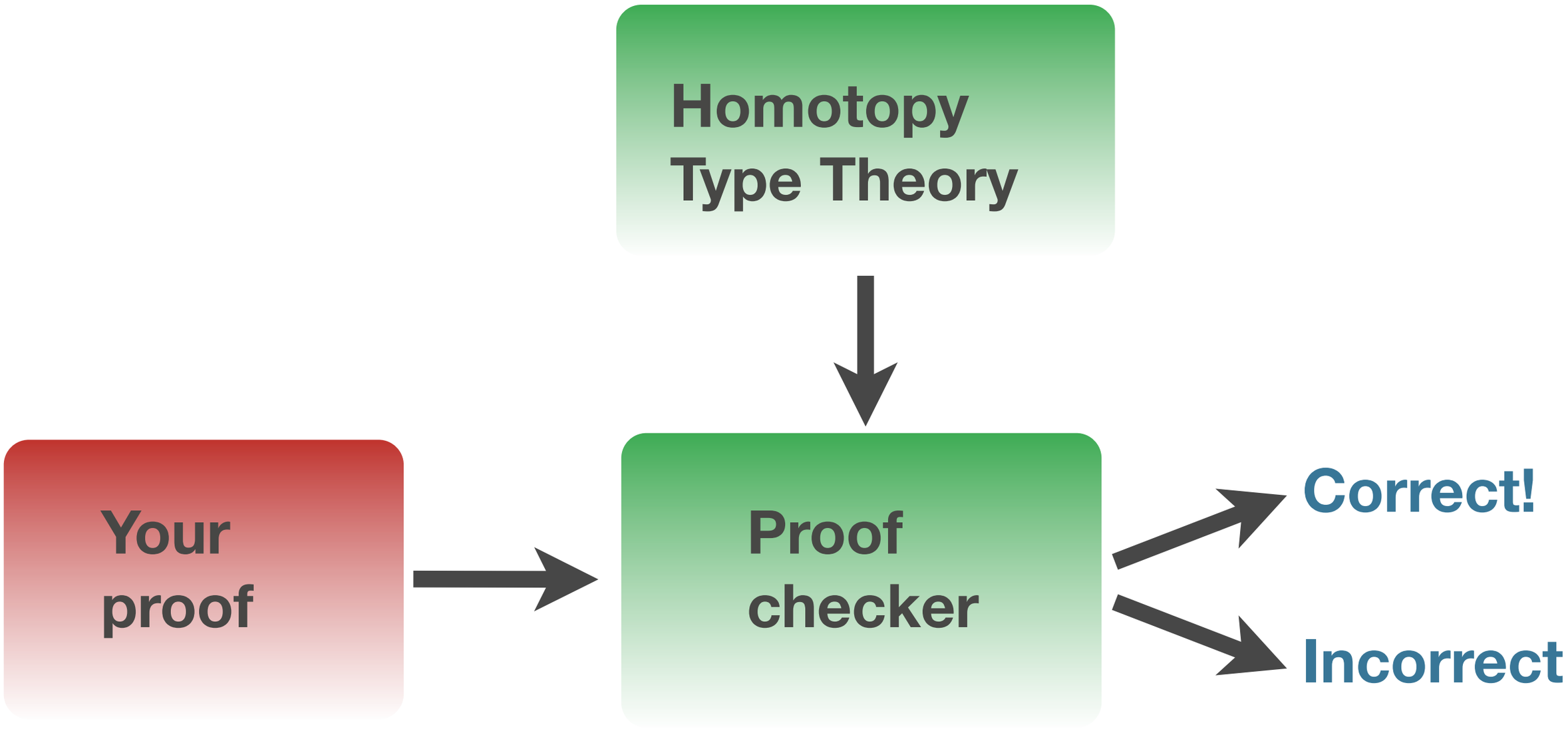
- >350,000 lines of math + code **~5-10x longer**
- >2 years to run **~2000x slower**

We have some work to do!

Homotopy Type Theory



Homotopy Type Theory



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 ∈ 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 ∈ 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 can have a non-trivial `<proof>`

Proof-relevance

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

Proof-relevance

type theory	set theory
<program> : <type>	x ∈ S
<proof> : <prog$_1$> = <prog$_2$>	x = y
<2-proof> : <proof$_1$> = <proof$_2$>	

Proof-relevance

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

⋮

Proof-relevance

type theory	**set theory**
$\langle program \rangle : \langle type \rangle$	$x \in S$
$\langle proof \rangle : \langle prog_1 \rangle = \langle prog_2 \rangle$	$x = y$
$\langle 2\text{-}proof \rangle : \langle proof_1 \rangle = \langle proof_2 \rangle$	
⋮	

Can have multiple different proofs of the same equality

Homotopy Theory

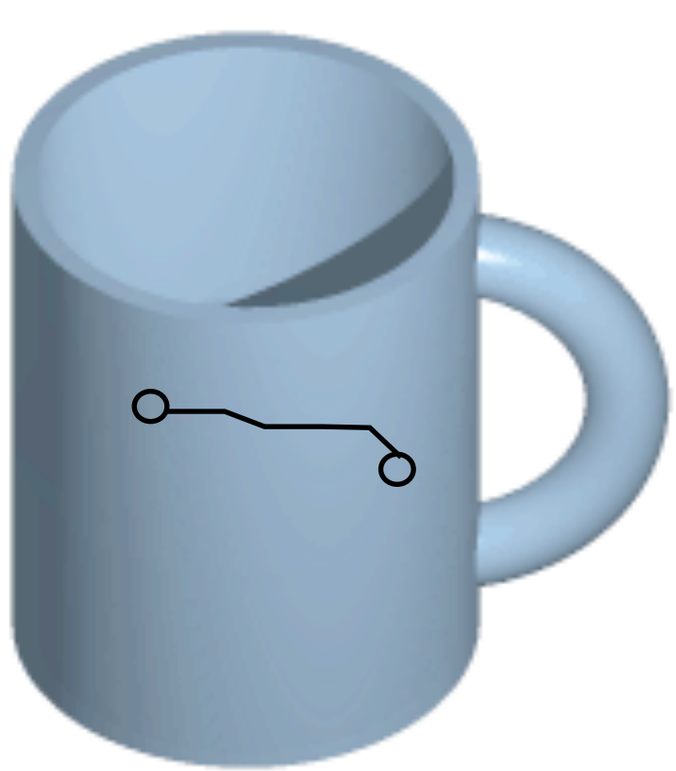
Homotopy Theory

Started as branch of topology,
the study of spaces and continuous deformations

[image from wikipedia]

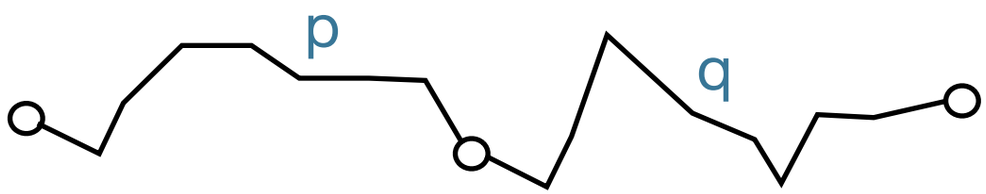
Homotopy Theory

Started as branch of topology,
the study of spaces and continuous deformations



[image from wikipedia]

Path Operations



Given paths p and $q : [0,1] \to X$ where $p(1) = q(0)$ define **composition** by:

$$(q \circ p)(x) = \begin{cases} p(2x) & \text{if } 0 \le x \le 1/2 \\ q(2x - 1) & \text{if } 1/2 \le x \le 1 \end{cases}$$

Homotopy

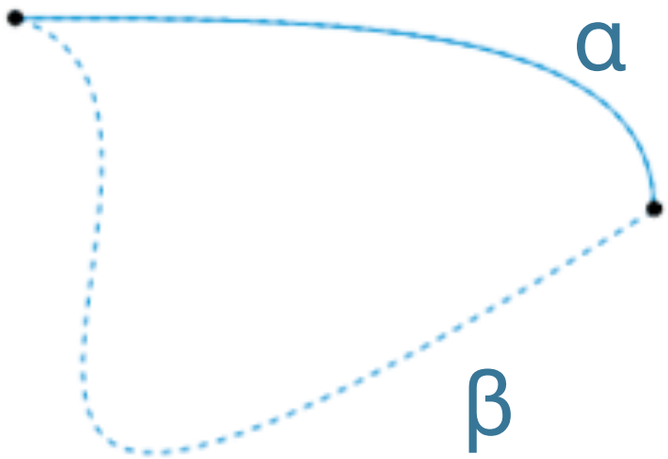
Deformation of one path into another

α

β

Homotopy

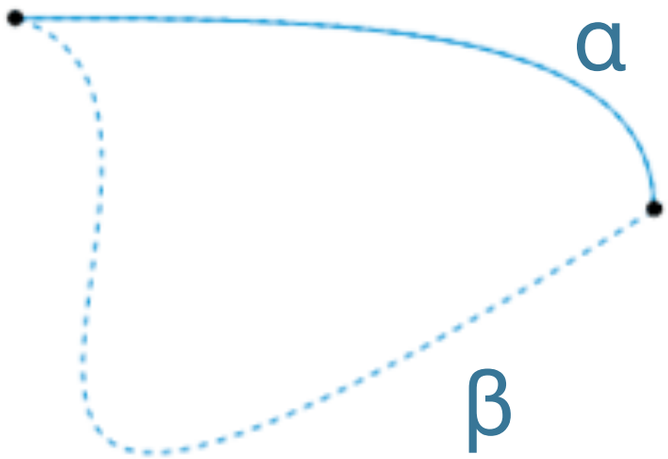
Deformation of one path into another



[image from wikipedia]

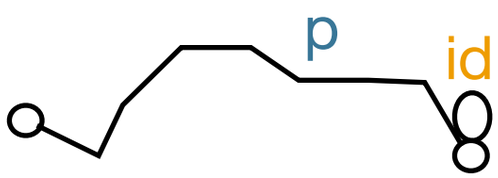
Homotopy

Deformation of one path into another

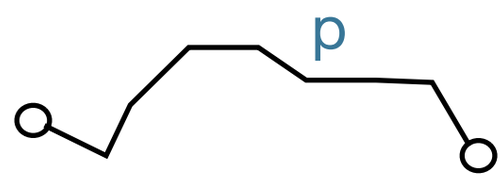


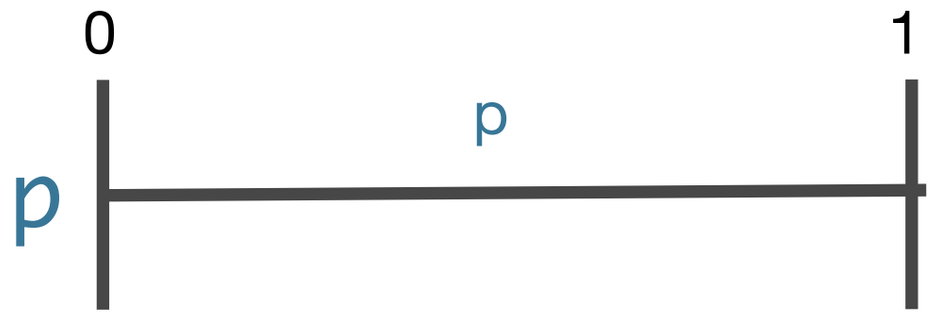
[image from wikipedia]

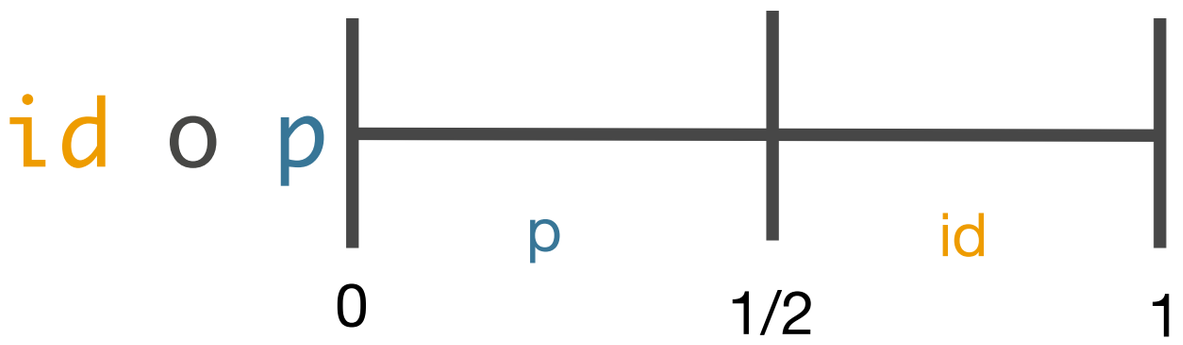
Unit law



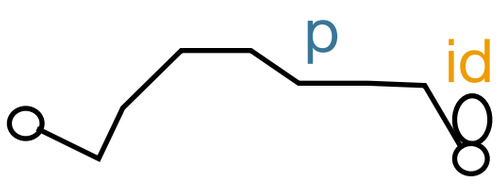
homotopic to



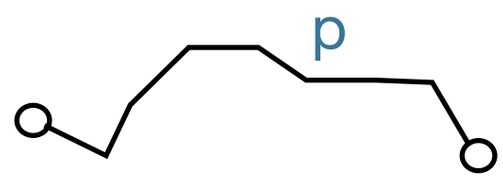


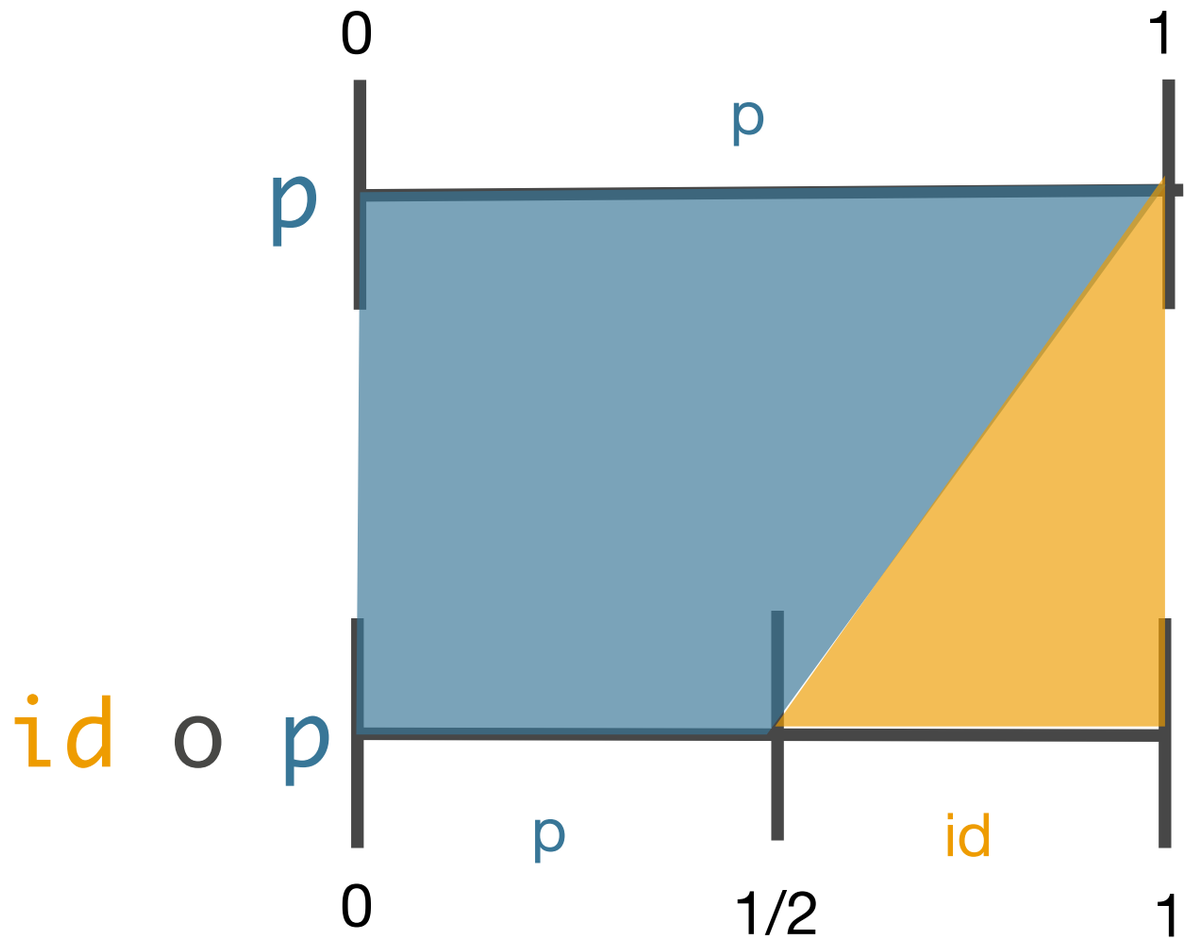


Unit law

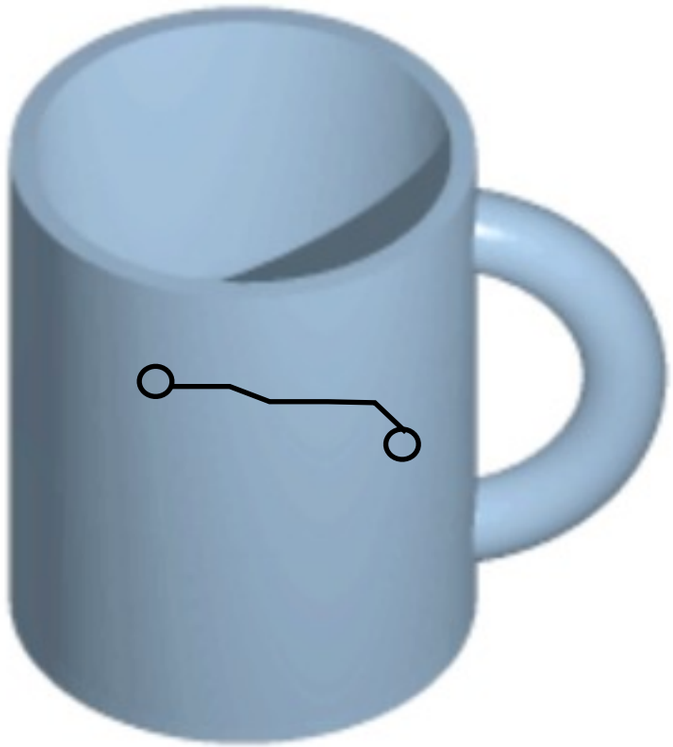


homotopic to



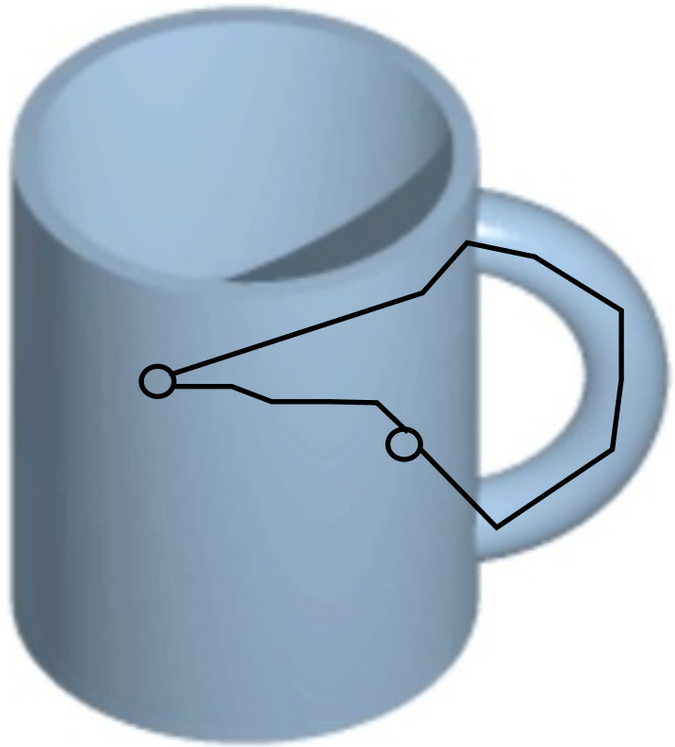


Non-homotopic paths



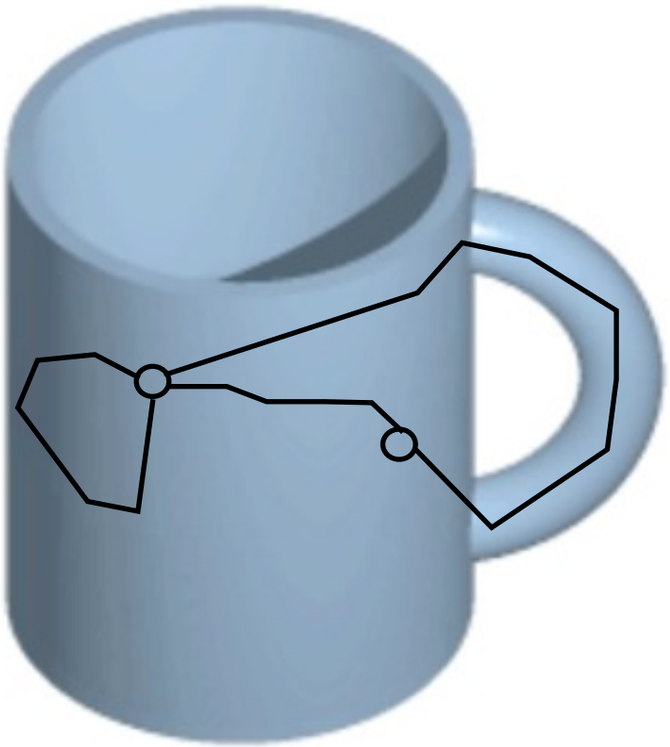
[image from wikipedia]

Non-homotopic paths



[image from wikipedia]

Non-homotopic paths



[image from wikipedia]

Homotopy type theory

Proofs of equality

reflexivity : M = M

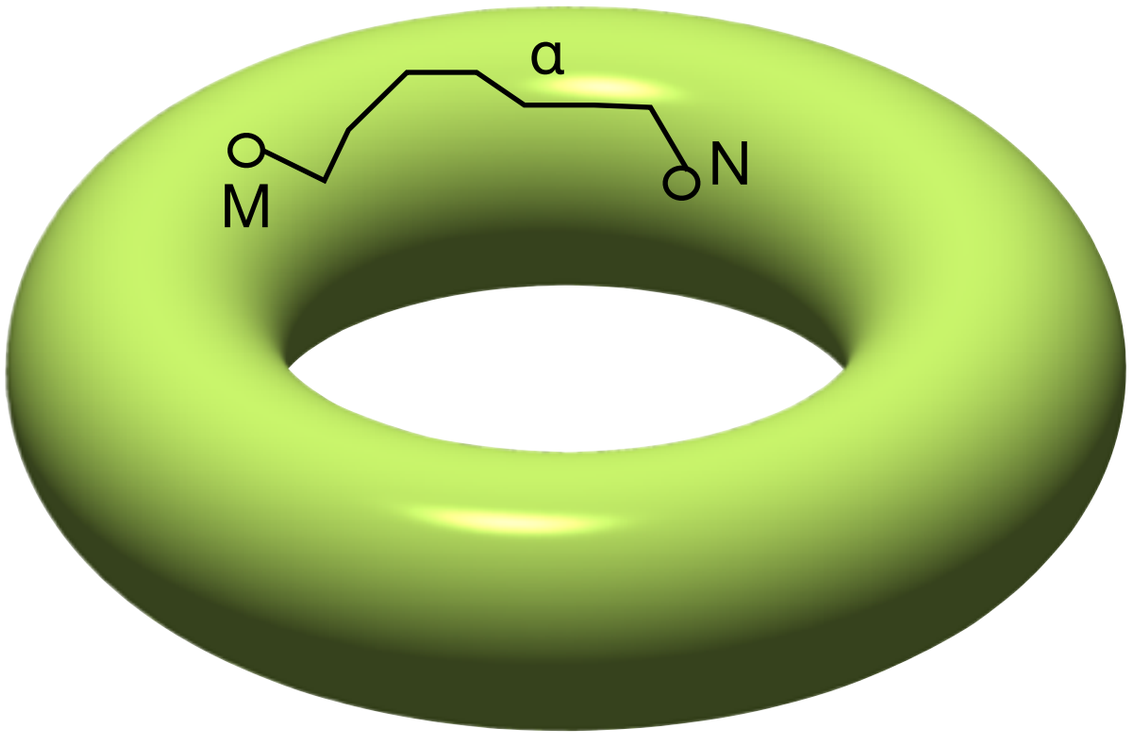
symmetry : N = M if M = N

transitivity : M = P if M = N and N = P

congruence : f(M) = f(N) if M = N

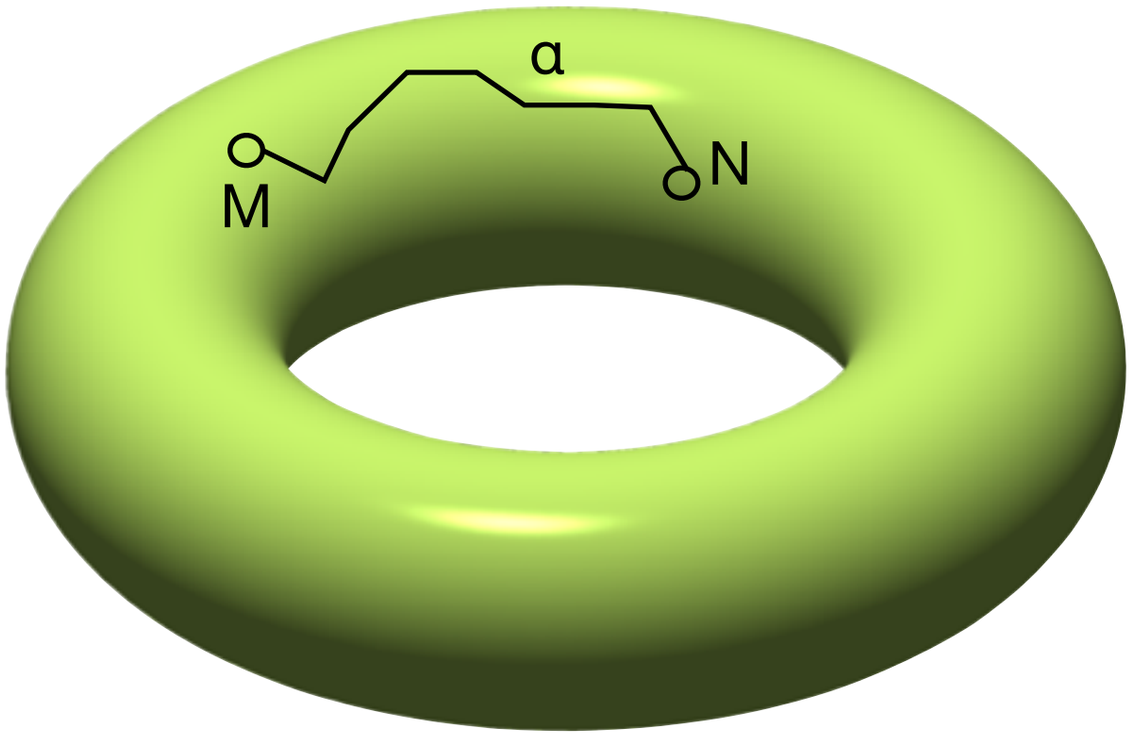
(plus *computation*)

Spaces as types



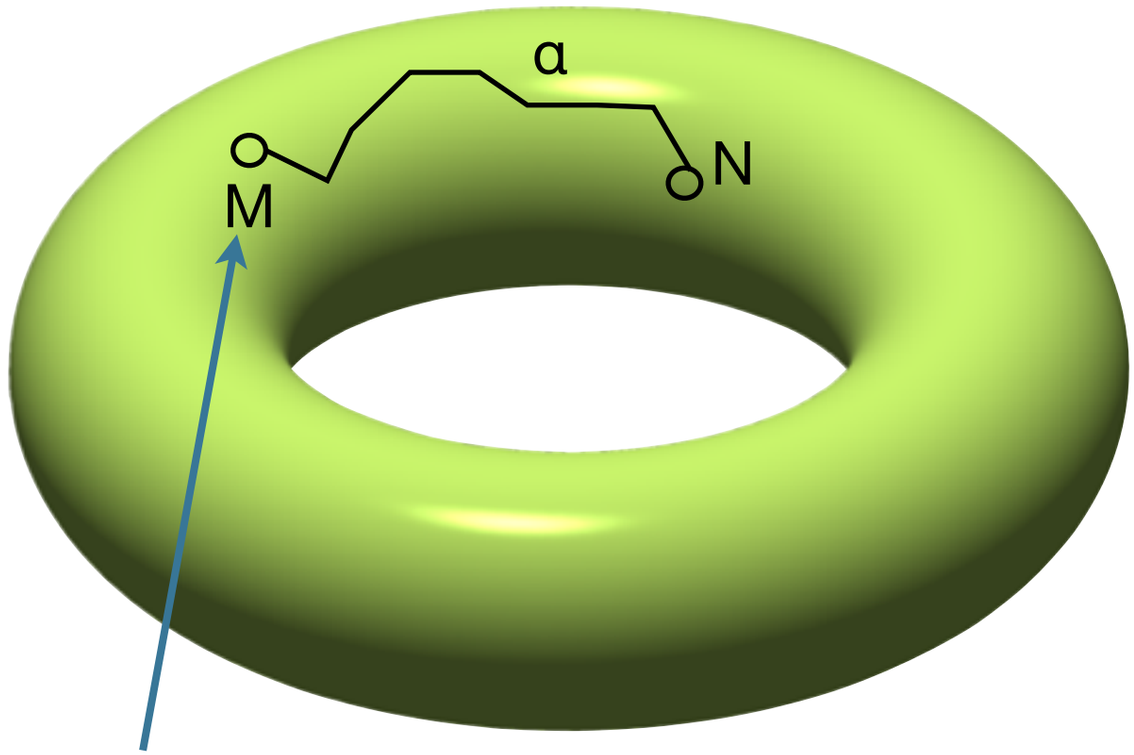
Spaces as types

a space is a type A



Spaces as types

a space is a type A

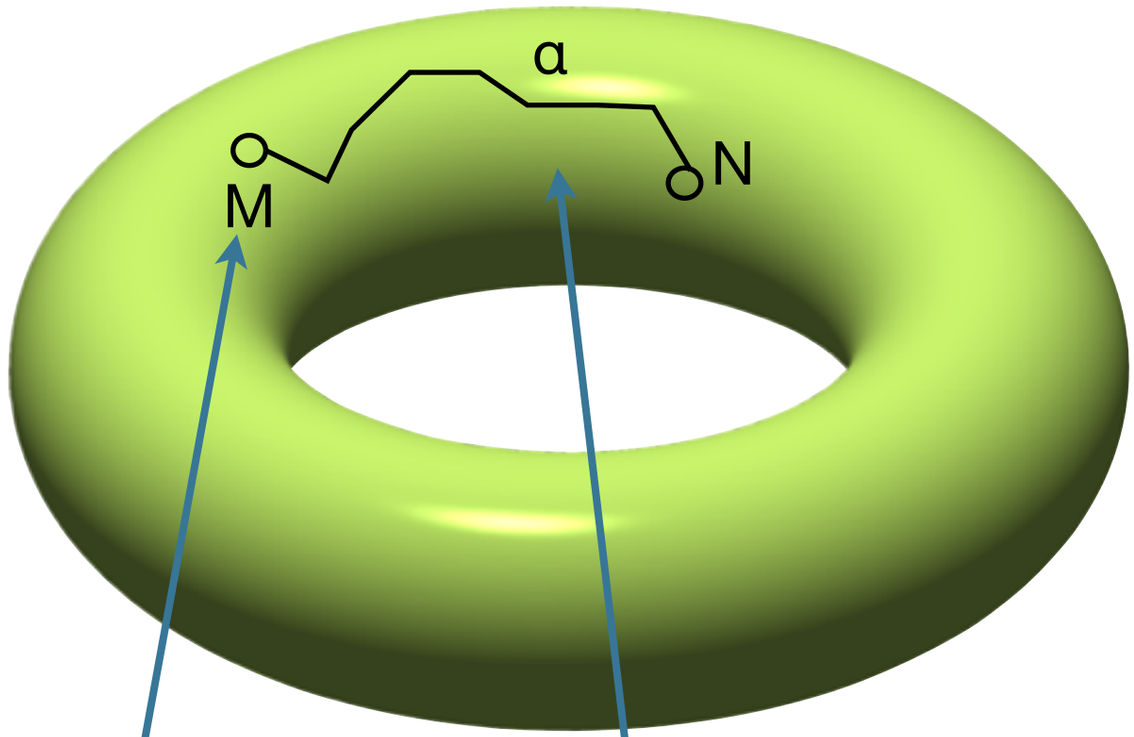


points are programs

M : A

Spaces as types

a space is a type A



points are programs

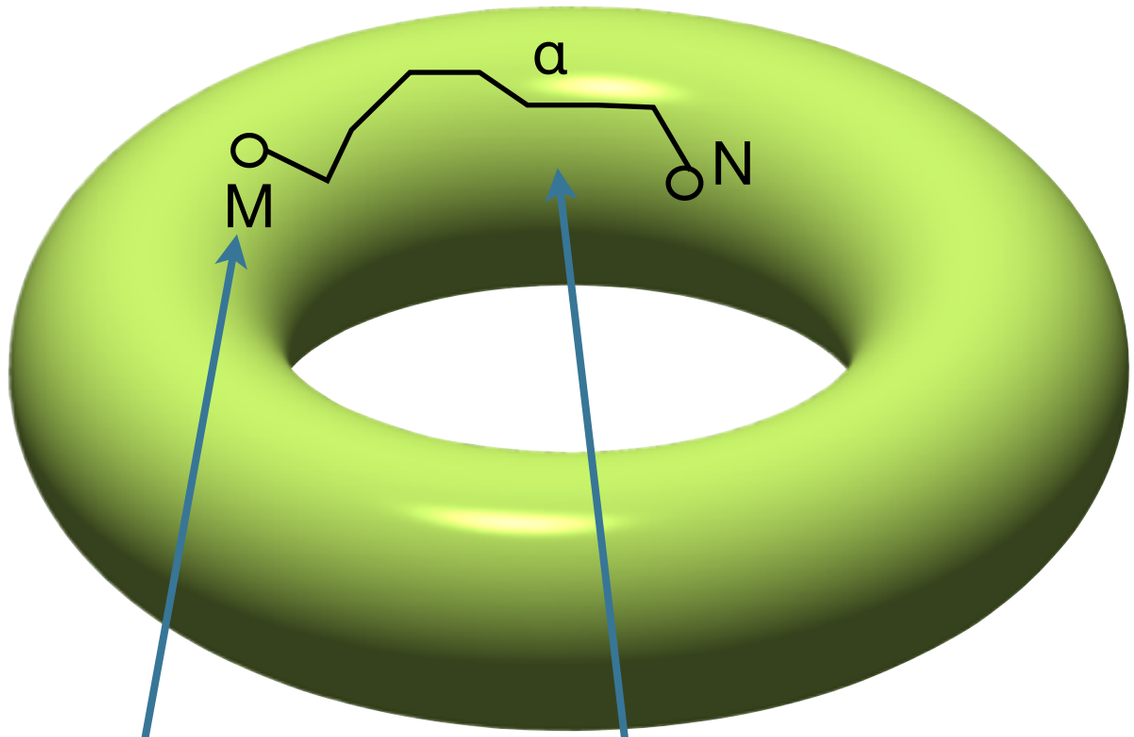
M : A

paths are *proofs of equality*

$\alpha : M =_A N$

Spaces as types

a space is a type A



points are programs

$M : A$

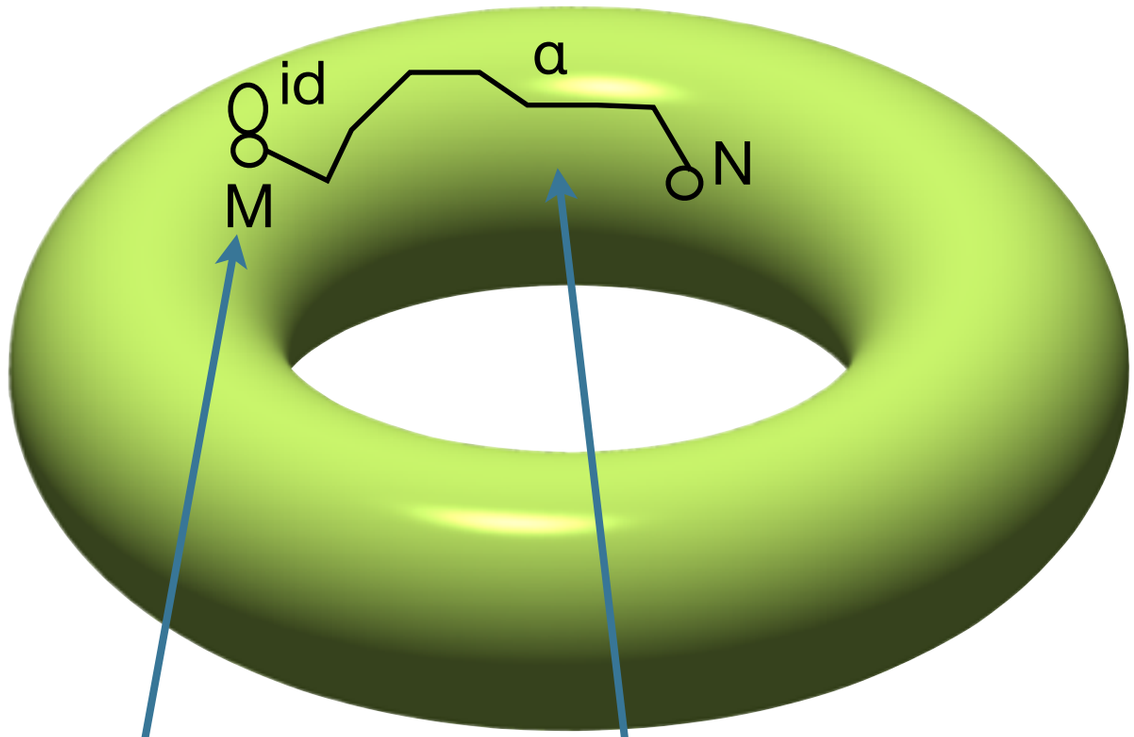
paths are *proofs of equality*

$\alpha \; : \; M =_A N$

path operations

Spaces as types

a space is a type A



points are programs

M : A

paths are *proofs of equality*

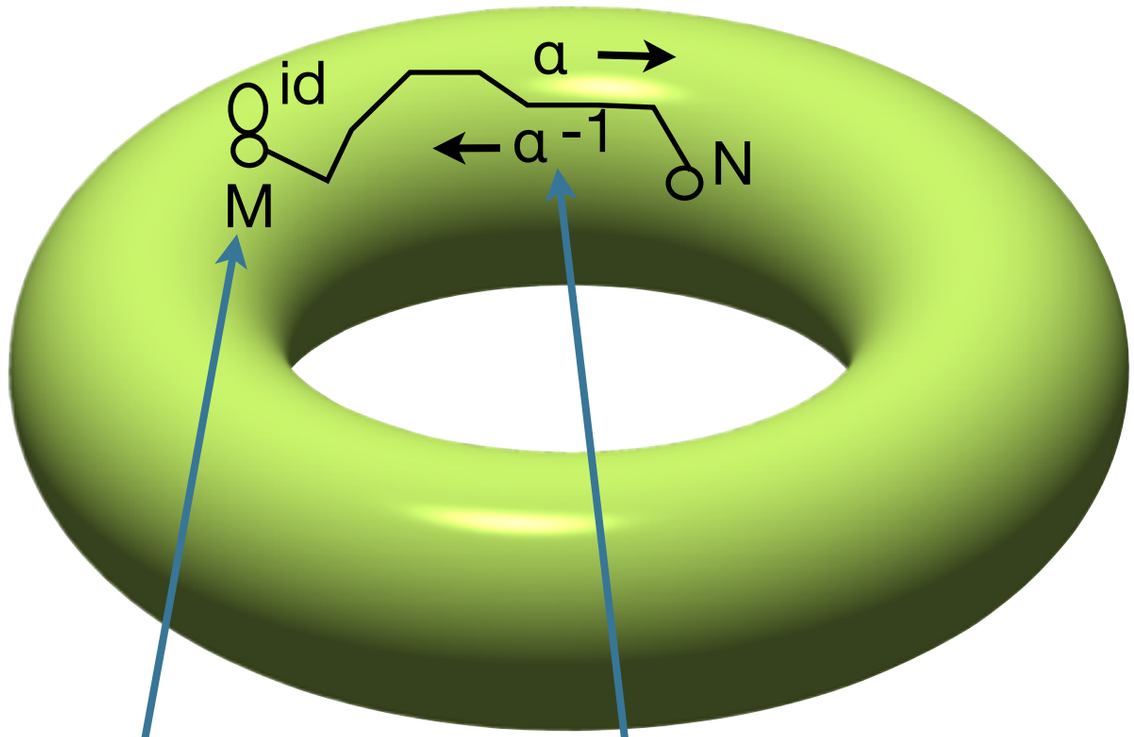
$\alpha : M =_A N$

path operations

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

Spaces as types

a space is a type A



points are programs

M : A

paths are *proofs of equality*

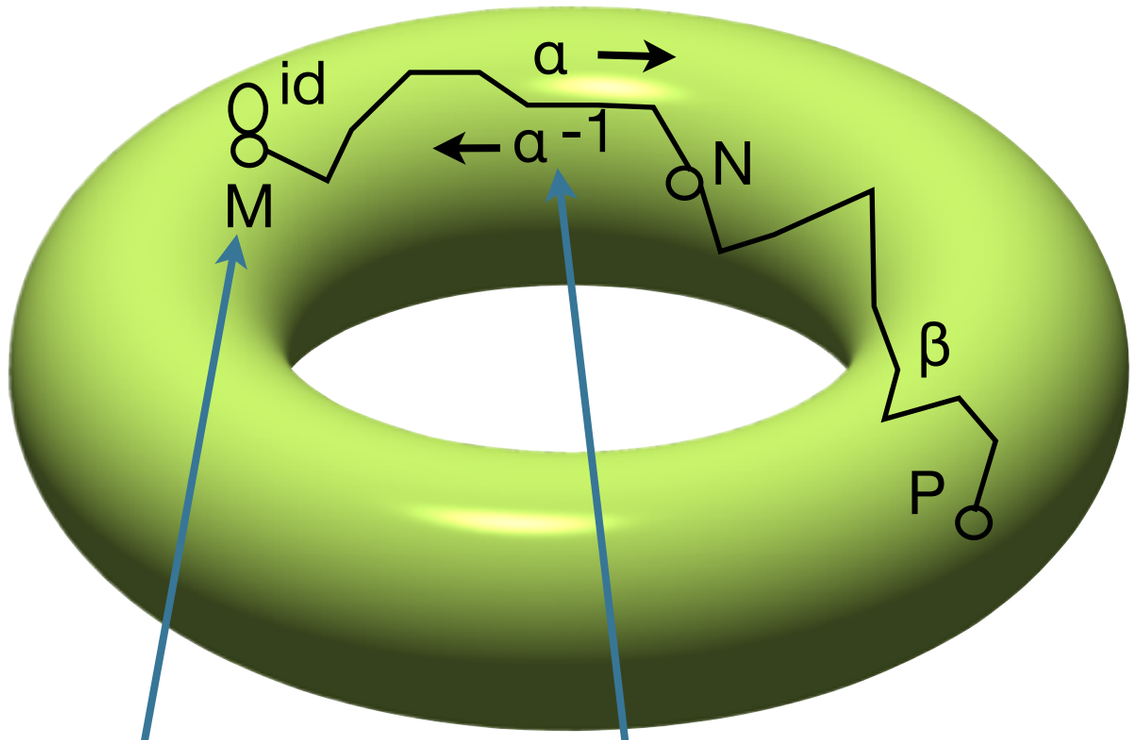
$\alpha : M =_A N$

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*

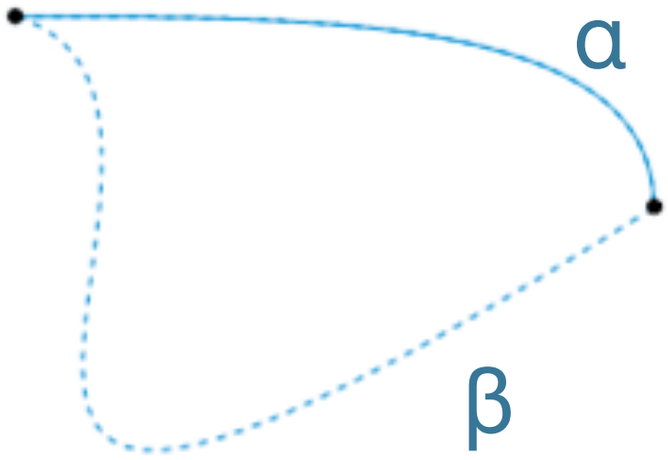
$\alpha : M =_A N$

path operations

```
id      : M = M (refl)
α⁻¹     : N = M (sym)
β o α   : M = P (trans)
```

Homotopy

Deformation of one path into another

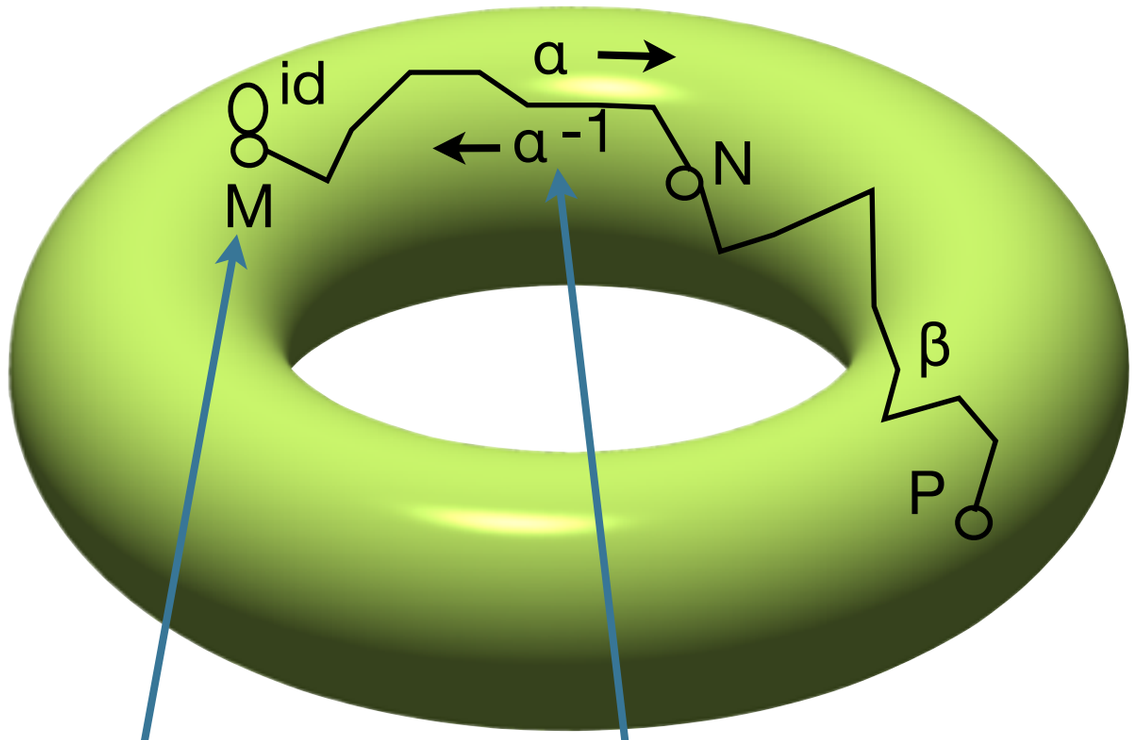


<2-proof> : α = β

[image from wikipedia]

Spaces as types

a space is a type A



points are programs

M : A

paths are *proofs of equality*

$\alpha : M =_A N$

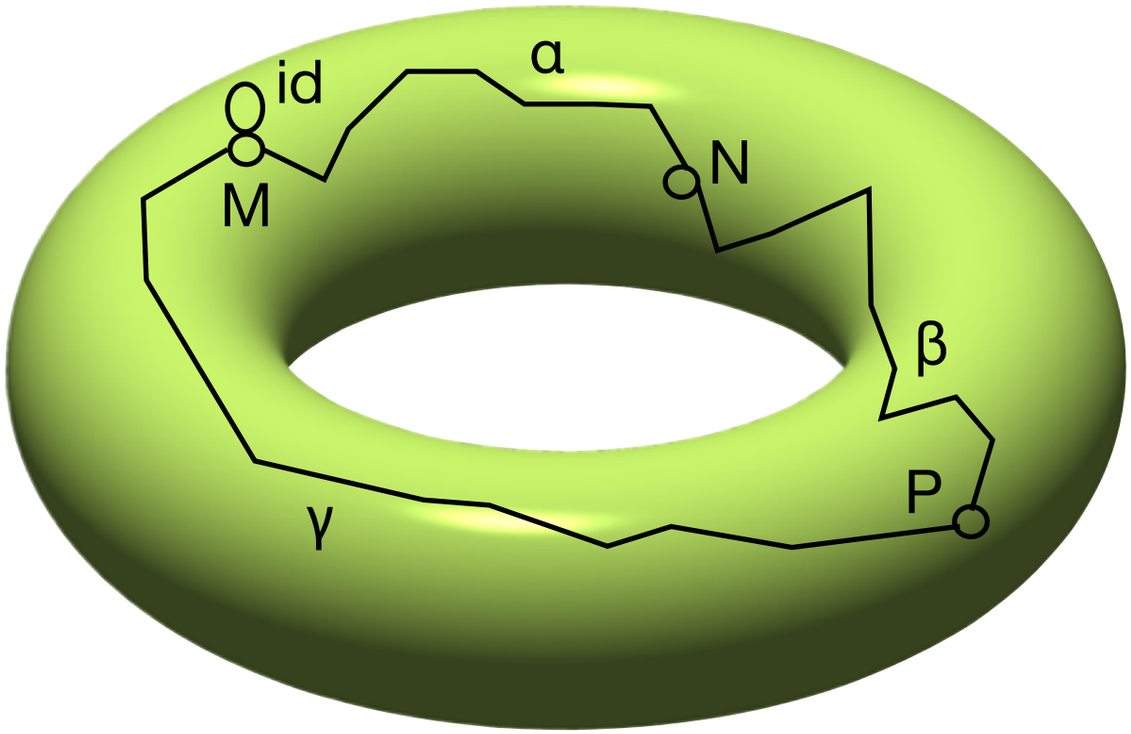
path operations

```
id      : M = M (refl)
α⁻¹     : N = M (sym)
β o α   : M = P (trans)
```

homotopies

```
id o p = p
p⁻¹ o p = id
r o (q o p)
   = (r o q) o p
```

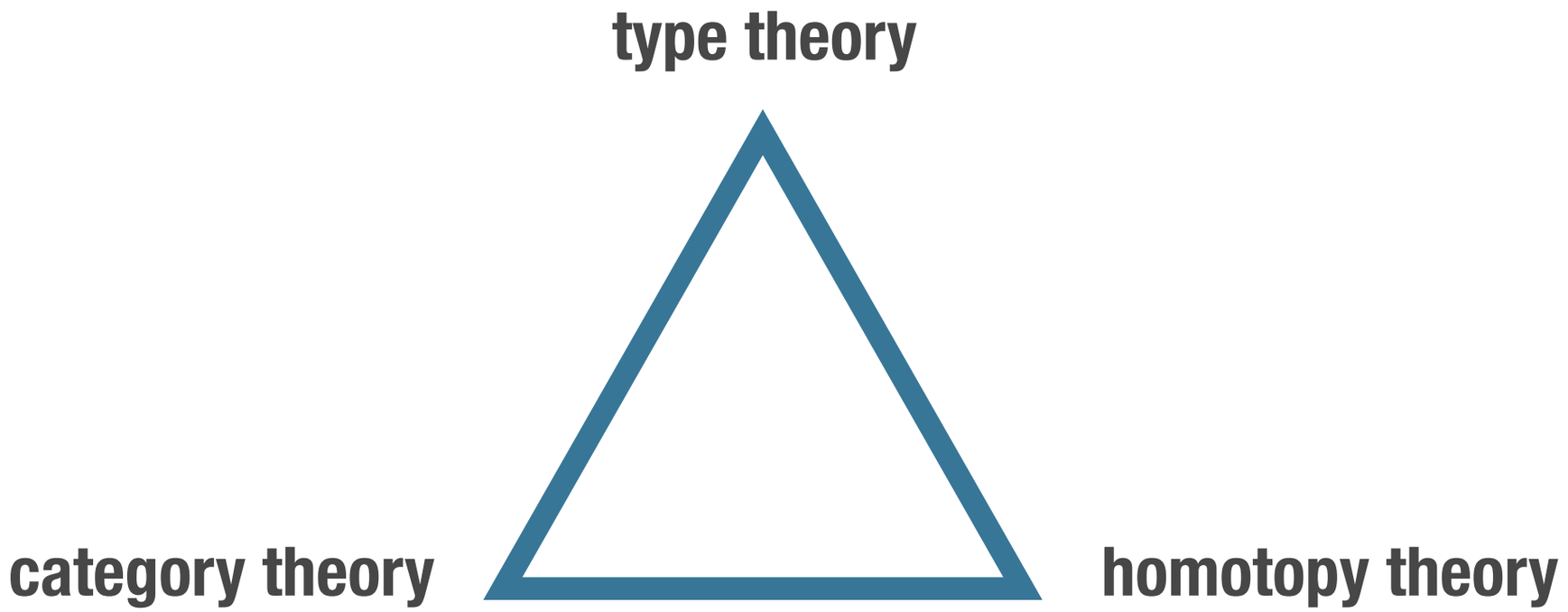
Different paths/proofs



γ o β o α != id

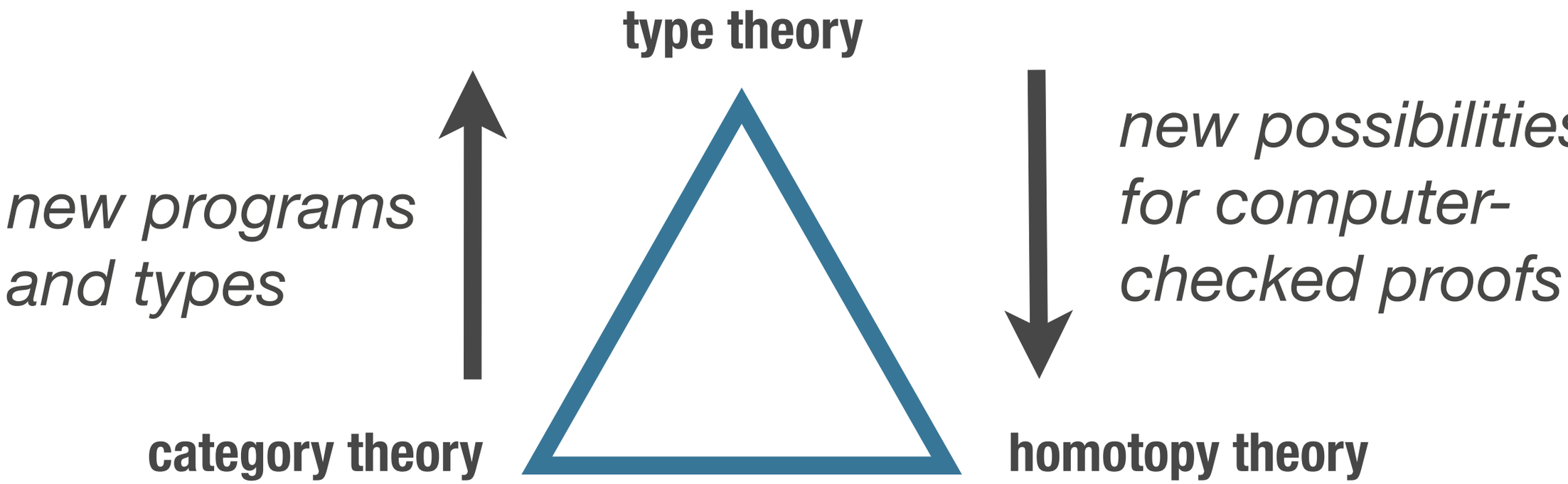
as proofs of M = M

Homotopy Type Theory



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

Homotopy Type Theory



Computer-checked homotopy theory

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_2$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Blakers-Massey

$T^2 = S^1 \times S^1$

Van Kampen

Covering spaces

Whitehead for n-types

Cohomology axioms

Mayer-Vietoris

[Brunerie, Cavallo, Finster, Hou, Licata, Lumsdaine, Shulman]

Homotopy groups of spheres

k^{th} homotopy group

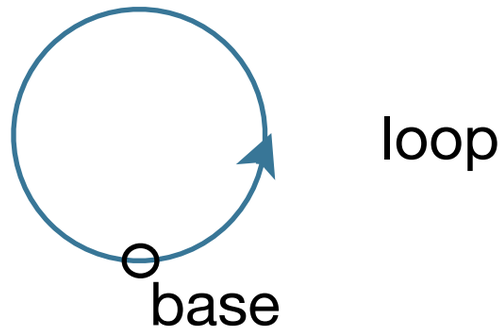
n-dimensional sphere

	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	$\mathbb{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
S^3	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
S^4	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{12}$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24}\times\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{120}\times\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^5$
S^5	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{72}\times\mathbb{Z}_2$
S^6	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_{24}\times\mathbb{Z}_2$	$\mathbb{Z}_2^3$
S^7	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_{120}$	$\mathbb{Z}_2^3$
S^8	0	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{120}$

[image from wikipedia]

The Circle

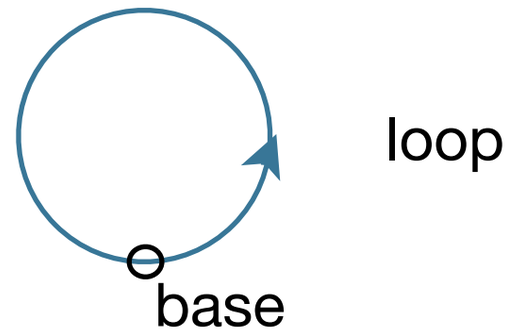
`Circle` is generated by



The Circle

`Circle` is generated by

```
base : Circle
loop : base = base
```

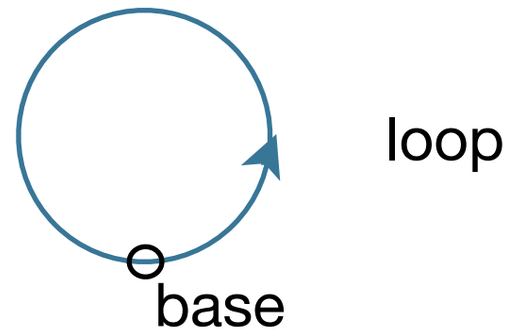


The Circle

`Circle` is generated by

point `base : Circle`

`loop : base = base`

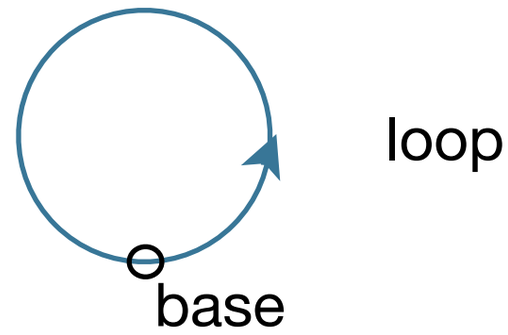


The Circle

`Circle` is generated by

point `base : Circle`

path `loop : base = base`

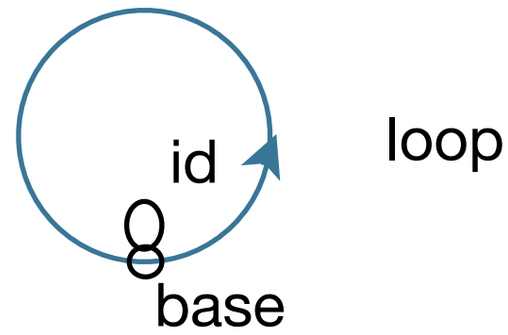


The Circle

`Circle` is generated by

point `base : Circle`

path `loop : base = base`



Also have the path operations, homotopies:

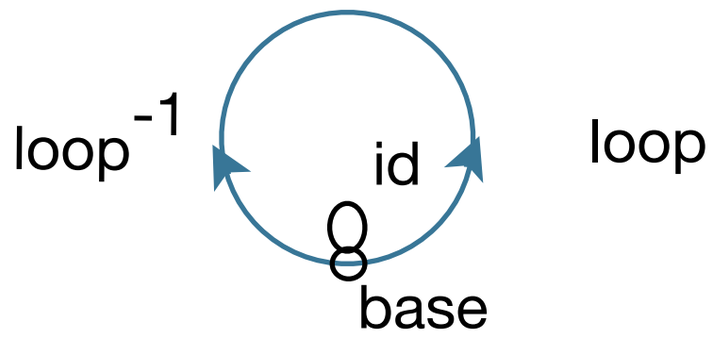
`id`

The Circle

`Circle` is generated by

point `base : Circle`

path `loop : base = base`



Also have the path operations, homotopies:

`id`

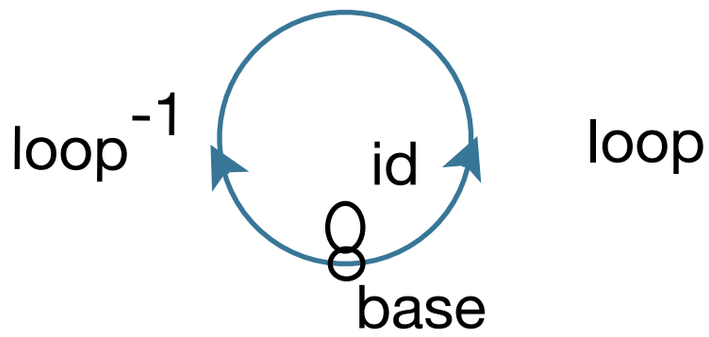
`loop`$^{-1}$

The Circle

`Circle` is generated by

point `base : Circle`

path `loop : base = base`



Also have the path operations, homotopies:

`id`

$\texttt{loop}^{-1}$

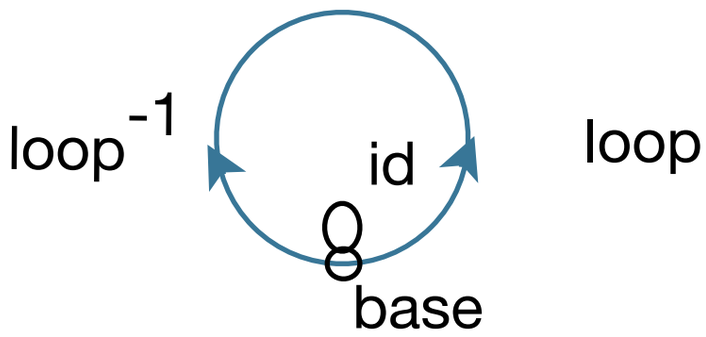
`loop o loop`

The Circle

`Circle` is generated by

point `base : Circle`

path `loop : base = base`



Also have the path operations, homotopies:

`id`

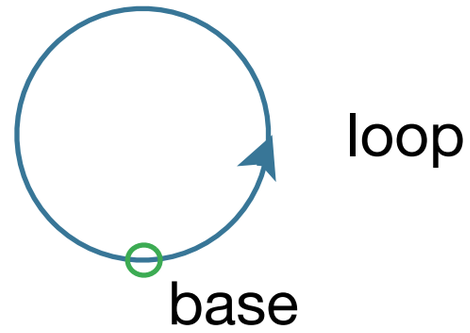
`loop^-1`

`loop o loop`

`inv : loop o loop^-1 = id`

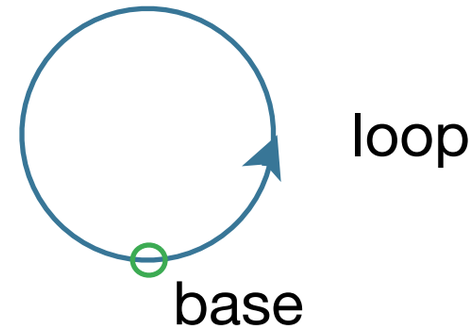
`...`

Counting paths



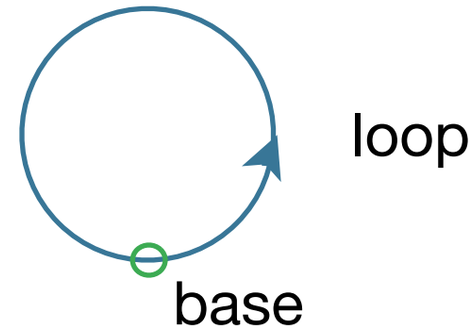
Counting paths

```
id
```



Counting paths

```
id
loop
```

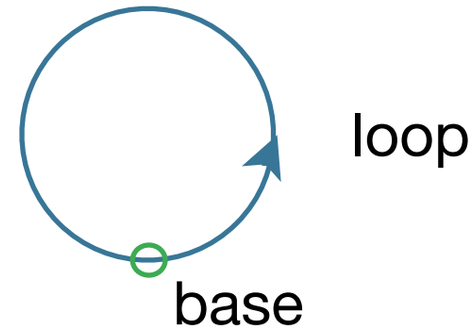


Counting paths

`id`

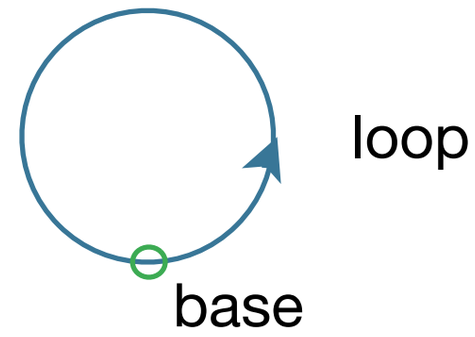
`loop`

$\texttt{loop}^{-1}$



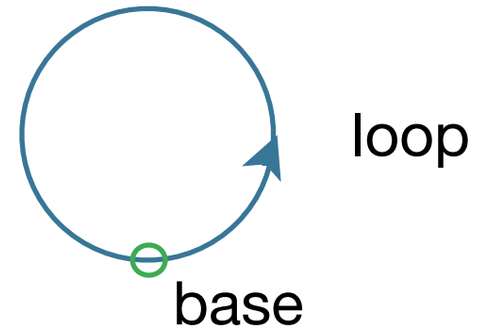
Counting paths

```
id
loop
loop⁻¹
loop o loop
```



Counting paths

```
id
loop
loop⁻¹
loop o loop
loop⁻¹ o loop⁻¹
```



Counting paths

id

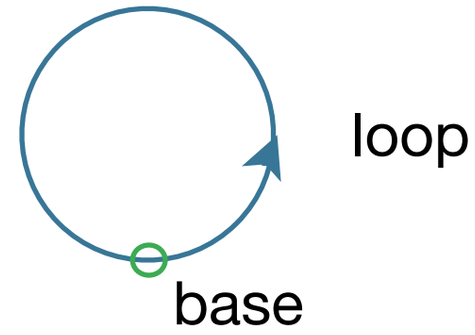
$loop$

$loop^{-1}$

$loop \circ loop$

$loop^{-1} \circ loop^{-1}$

$loop \circ loop^{-1}$



Counting paths

id

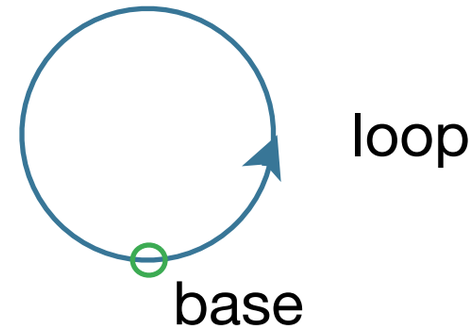
$loop$

$loop^{-1}$

$loop \circ loop$

$loop^{-1} \circ loop^{-1}$

$loop \circ loop^{-1} = id$



Counting paths

id 0

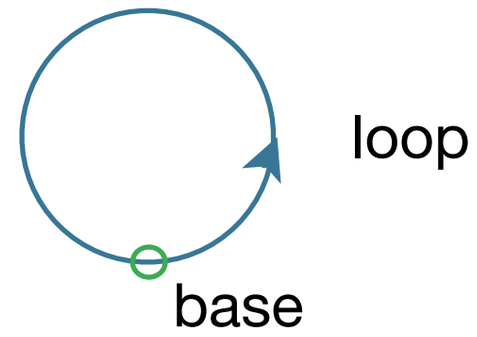
$loop$

$loop^{-1}$

$loop \circ loop$

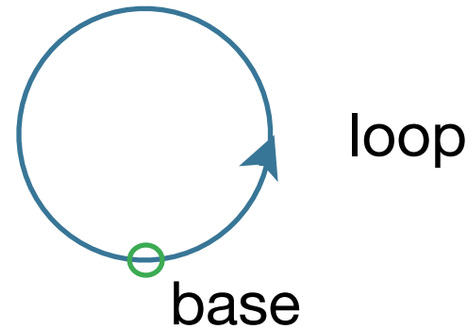
$loop^{-1} \circ loop^{-1}$

$loop \circ loop^{-1} = id$



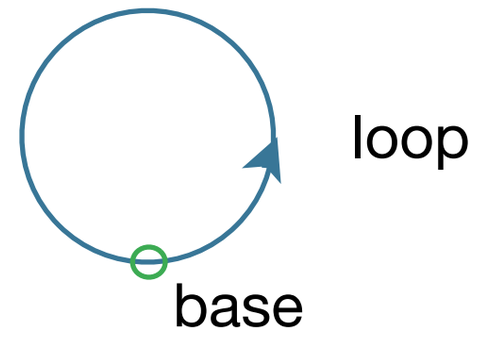
Counting paths

id	0
$loop$	1
$loop^{-1}$	
$loop \circ loop$	
$loop^{-1} \circ loop^{-1}$	
$loop \circ loop^{-1} = id$	



Counting paths

id	0
$loop$	1
$loop^{-1}$	-1



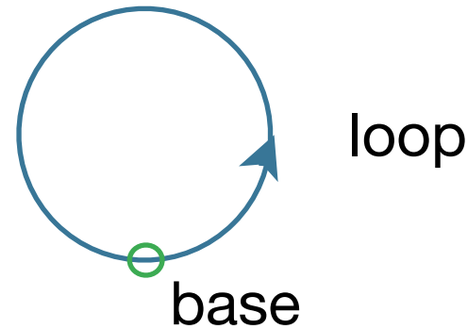
$$loop \circ loop$$

$$loop^{-1} \circ loop^{-1}$$

$$loop \circ loop^{-1} = id$$

Counting paths

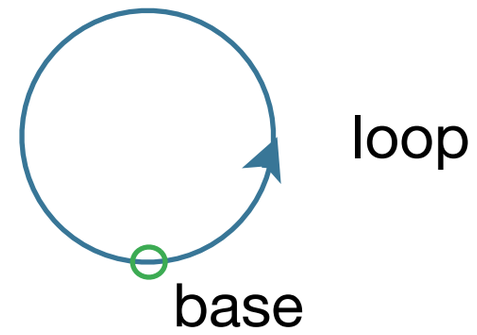
id	0
$loop$	1
$loop^{-1}$	-1
$loop \circ loop$	2
$loop^{-1} \circ loop^{-1}$	
$loop \circ loop^{-1} = id$	



Counting paths

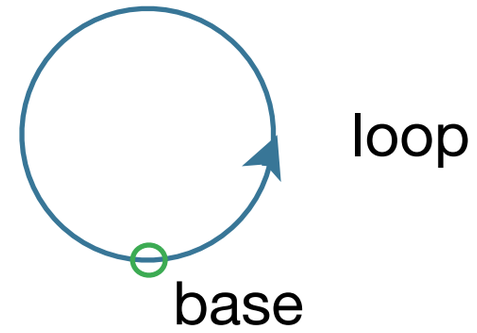
`id`	`0`
`loop`	`1`
`loop`$^{-1}$	`-1`
`loop o loop`	`2`
`loop`$^{-1}$ `o loop`$^{-1}$	`-2`

`loop o loop`$^{-1}$ `= id`



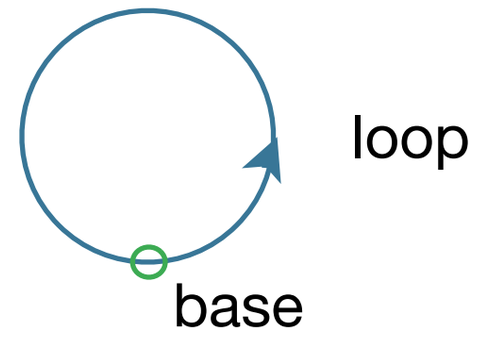
Counting paths

id	0
$loop$	1
$loop^{-1}$	-1
$loop \circ loop$	2
$loop^{-1} \circ loop^{-1}$	-2
$loop \circ loop^{-1} = id$	0



Counting paths

`id`	`0`
`loop`	`1`
`loop`$^{-1}$	`-1`
`loop o loop`	`2`
`loop`$^{-1}$ `o loop`$^{-1}$	`-2`
`loop o loop`$^{-1}$ `= id`	`0`



Theorem: that's it

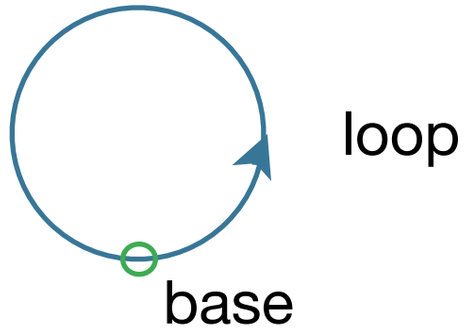
(Fundamental group of the circle is $\mathbb{Z}$)

Functions act on paths

congruence says f(M) = f(N) if M = N

Functions act on paths

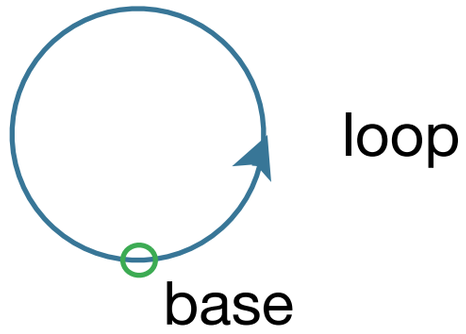
congruence says f(M) = f(N) if M = N



Functions act on paths

congruence says f(M) = f(N) if M = N

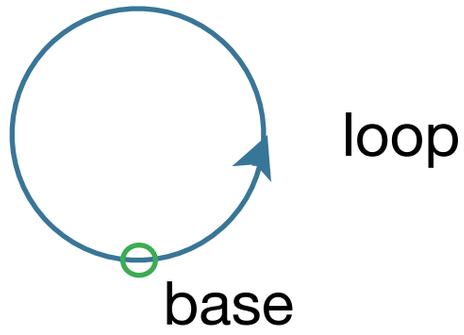
f(p) : f(M) = f(N) if p : M = N



Functions act on paths

congruence says f(M) = f(N) if M = N

f(p) : f(M) = f(N) if p : M = N

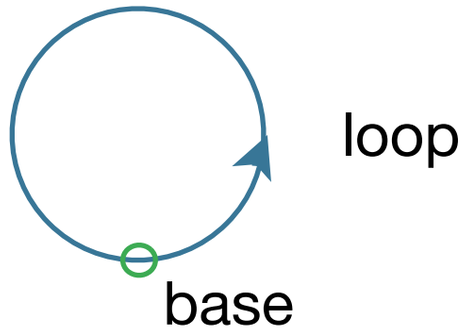



```
f : Circle → X
f(base) = …
f(loop) = … : f(base) = f(base)
```

Functions act on paths

congruence says f(M) = f(N) if M = N

f(p) : f(M) = f(N) if p : M = N

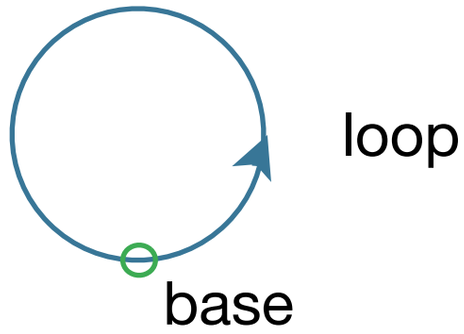



```
f : Circle → X
f(base) = …
f(loop) = … : f(base) = f(base)

f(id) =
f(loop o loop) =
f(loop⁻¹) =
```

Functions act on paths

congruence says f(M) = f(N) if M = N

f(p) : f(M) = f(N) if p : M = N

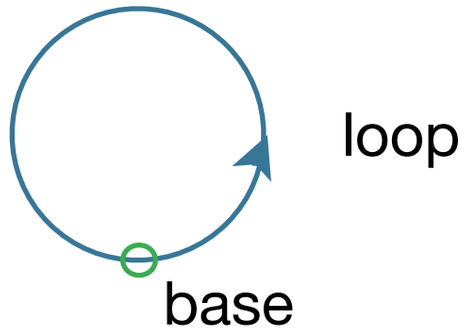



```
f : Circle → X
f(base) = …
f(loop) = … : f(base) = f(base)

f(id) = id
f(loop o loop) =
f(loop⁻¹) =
```

Functions act on paths

congruence says f(M) = f(N) if M = N

f(p) : f(M) = f(N) if p : M = N

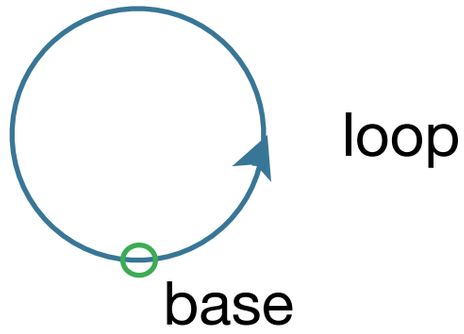



```
f : Circle → X
f(base) = …
f(loop) = … : f(base) = f(base)

f(id) = id
f(loop o loop) = f(loop) o f(loop)
f(loop^-1) =
```

Functions act on paths

congruence says f(M) = f(N) if M = N

f(p) : f(M) = f(N) if p : M = N



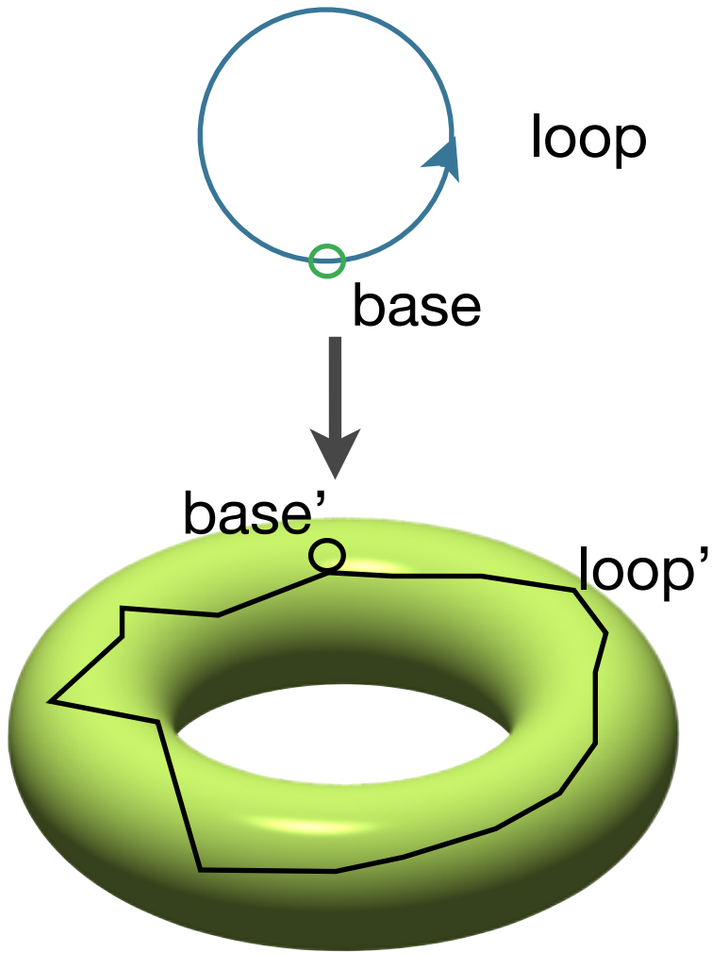

```
f : Circle → X
f(base) = …
f(loop) = … : f(base) = f(base)

f(id) = id
f(loop o loop) = f(loop) o f(loop)
f(loop⁻¹) = f(loop)⁻¹
```

Circle recursion

Circle recursion:

`f : Circle → X` determined by

```
f base = base'
f loop = loop'
    : f(base) = f(base')
```

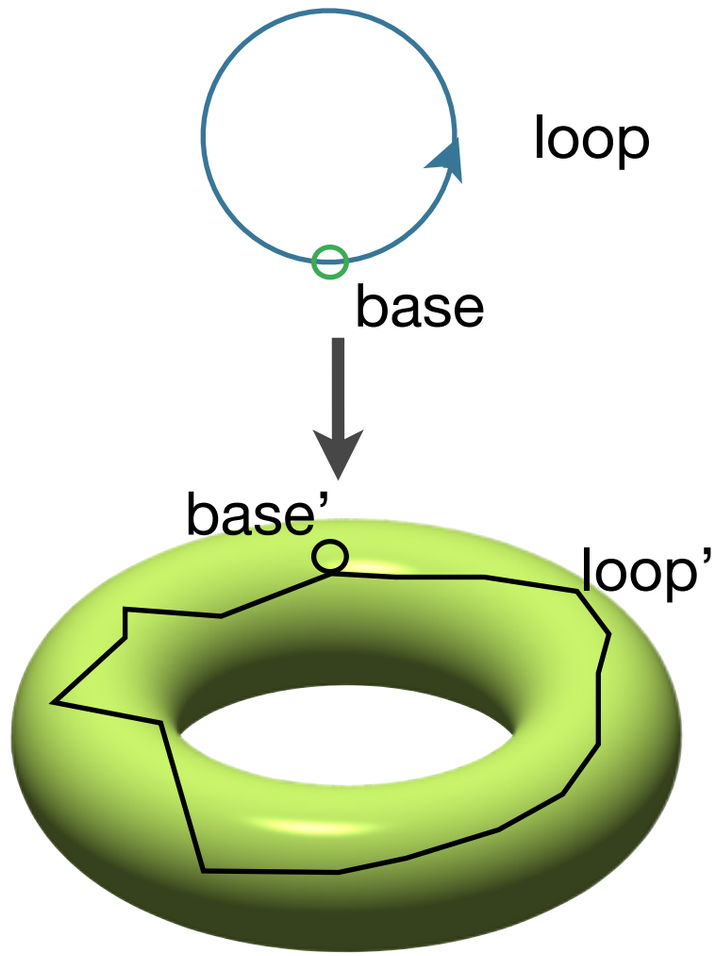


Circle recursion

Circle recursion:

`f : Circle → X` determined by

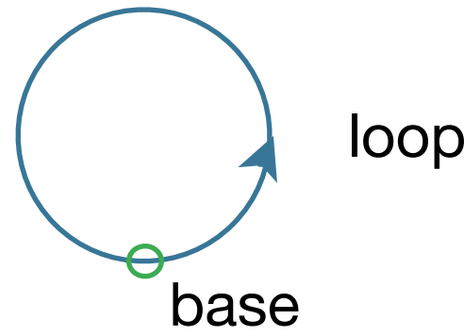
```
f base = base'
f loop = loop'
    : f(base) = f(base')
```



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

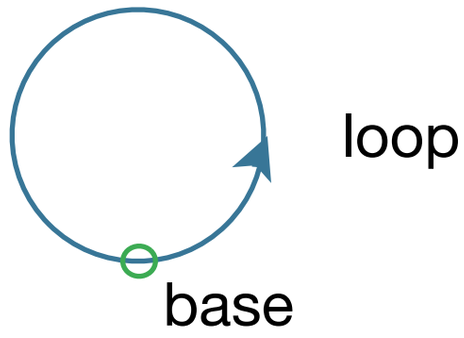
Circle recursion

```
speedup : Circle → Circle
speedup base = base
speedup loop = loop o loop
```



Circle recursion

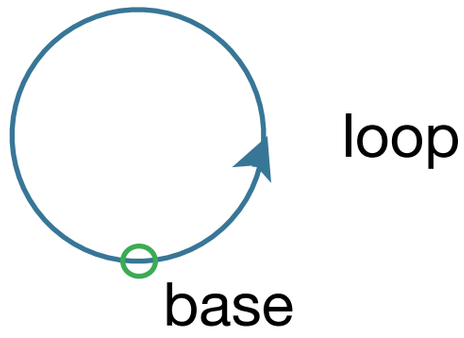
```
speedup : Circle → Circle
speedup base = base
speedup loop = loop o loop
```



It follows that:

Circle recursion

```
speedup : Circle → Circle
speedup base = base
speedup loop = loop o loop
```

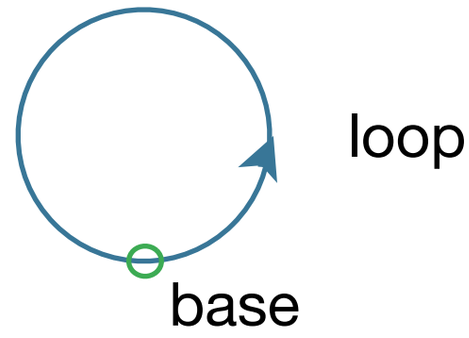


It follows that:

```
   speedup(loop o loop)
= speedup(loop) o speedup(loop)
= (loop o loop) o (loop o loop)
```

Circle recursion

```
speedup : Circle → Circle
speedup base = base
speedup loop = loop o loop
```



It follows that:

```
   speedup(loop o loop)
= speedup(loop) o speedup(loop)
= (loop o loop) o (loop o loop)

   speedup(loop^-1)
= (speedup(loop))^-1
= (loop o loop)^-1 = loop^-1 o loop^-1
```

The Circle (Take 2)

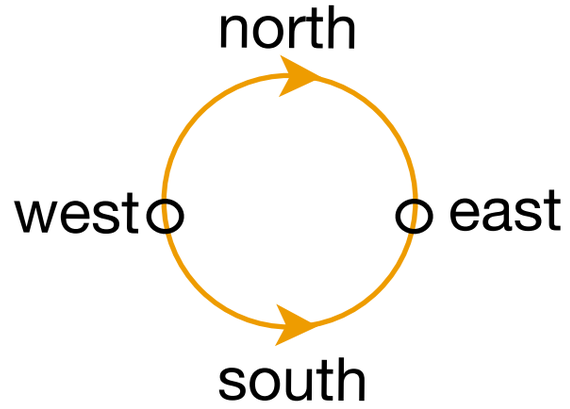
$\texttt{Circle}_2$ is generated by

point $\texttt{west} : \texttt{Circle}_2$

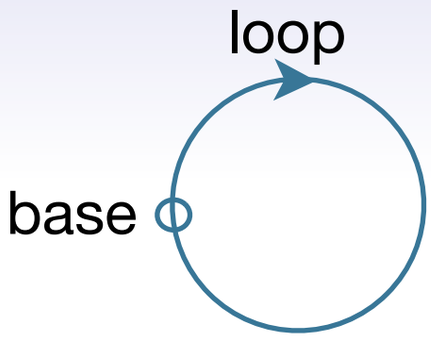
$\texttt{east} : \texttt{Circle}_2$

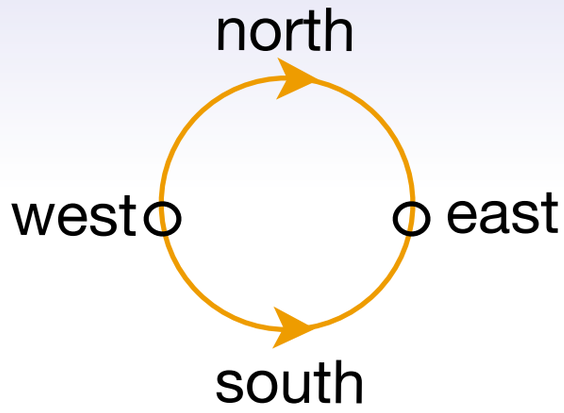
path $\texttt{north} : \texttt{west} = \texttt{east}$

$\texttt{south} : \texttt{west} = \texttt{east}$



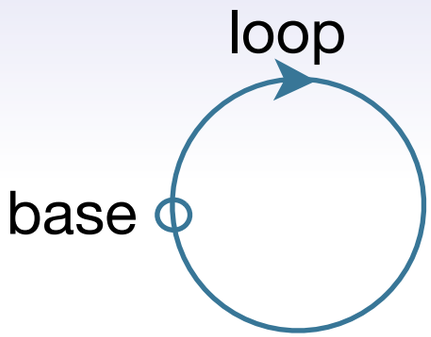
Theorem

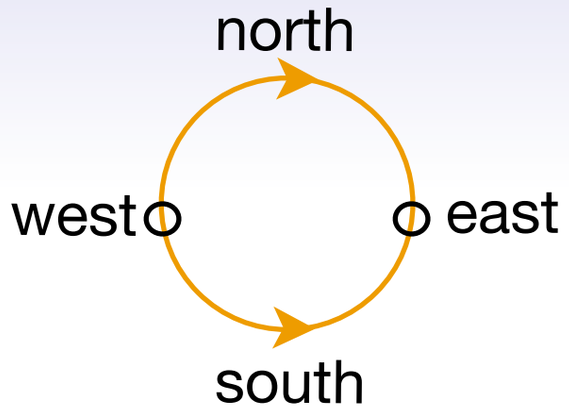




`Circle` and $\texttt{Circle}_2$ are *homotopy equivalent*

Theorem

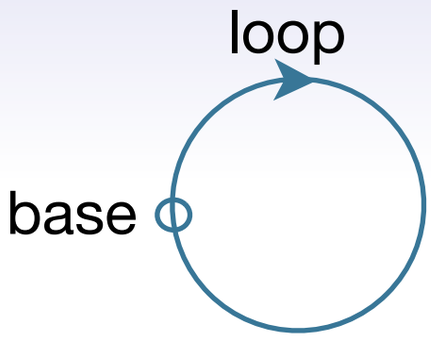


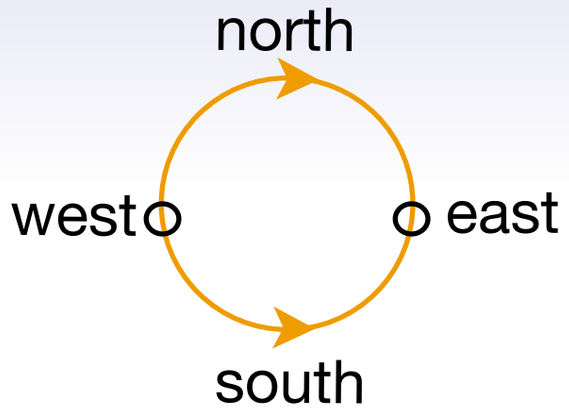


`Circle` and $\texttt{Circle}_2$ are *homotopy equivalent*

```
one2two  : Circle → Circle₂
two2one  : Circle₂ → Circle

∀x, one2two(two2one x) = x

∀y, two2one(one2two x) = x
```

Theorem

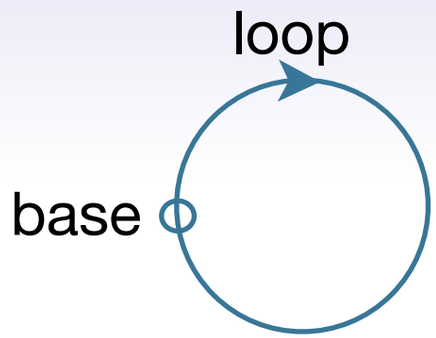


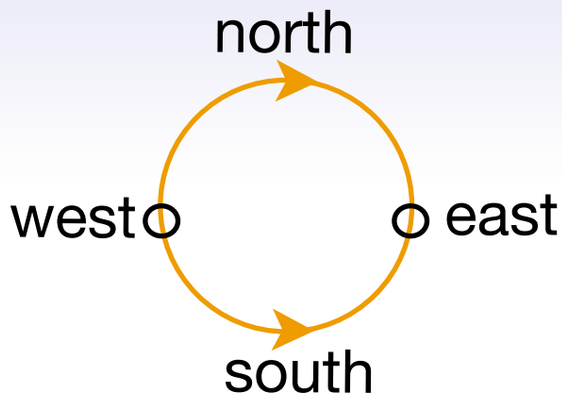


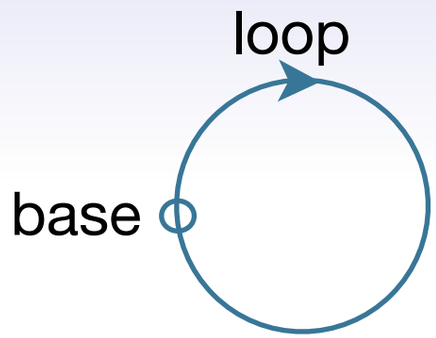
`Circle` and $\texttt{Circle}_2$ are *homotopy equivalent*

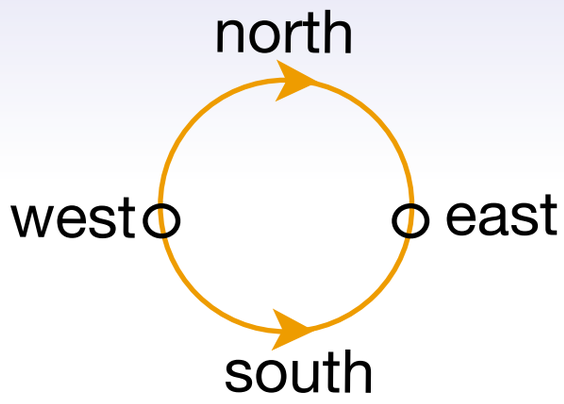
```
one2two  : Circle → Circle₂
two2one  : Circle₂ → Circle

∀x, one2two(two2one x) = x

∀y, two2one(one2two x) = x
```

really means path

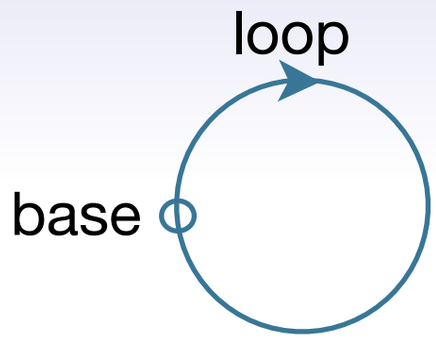

loop
base

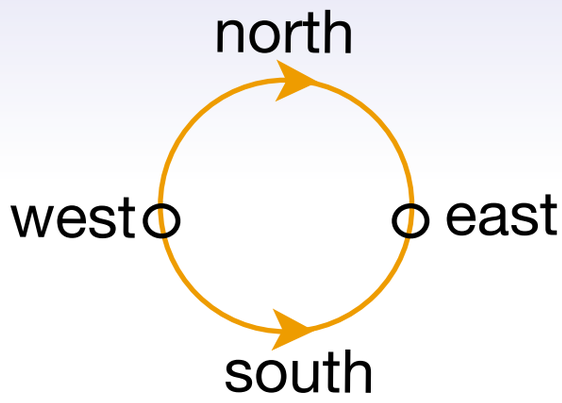

north
west
east
south

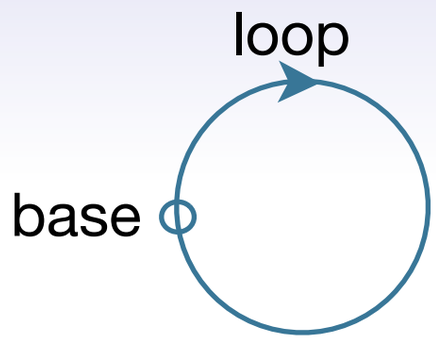


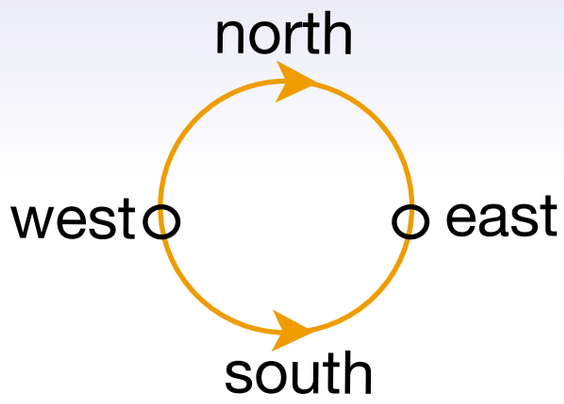



```
one2two  : Circle → Circle₂
one2two base = west
one2two loop = south⁻¹ ∘ north
```


loop
base

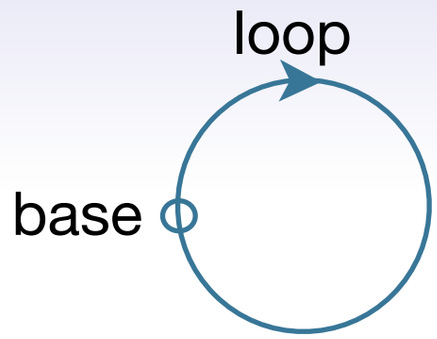

north
west
east
south

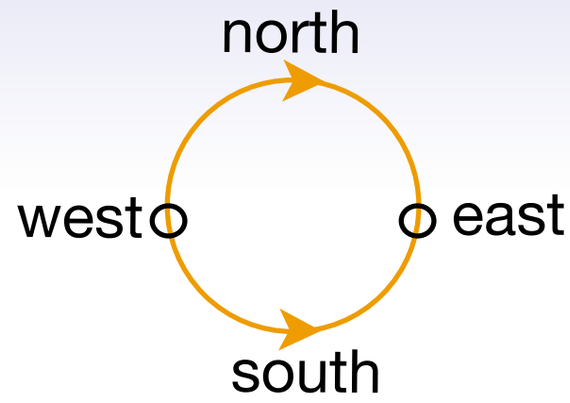





```
two2one  : Circle₂ → Circle
two2one west = base
two2one east = base
two2one north = loop
two2one south = id
```

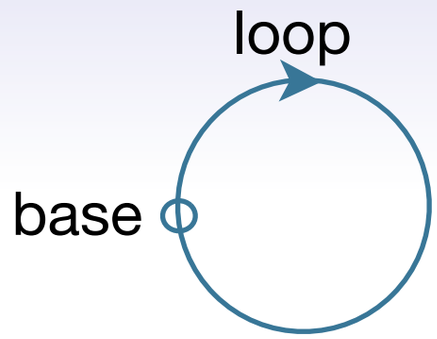
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```

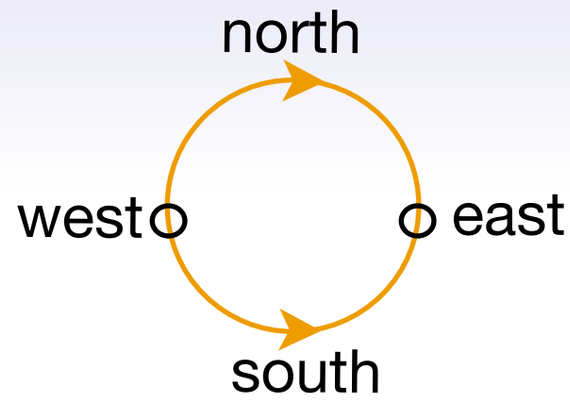




$$\forall x,\ \text{two2one}(\text{one2two}\ x) = x$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```

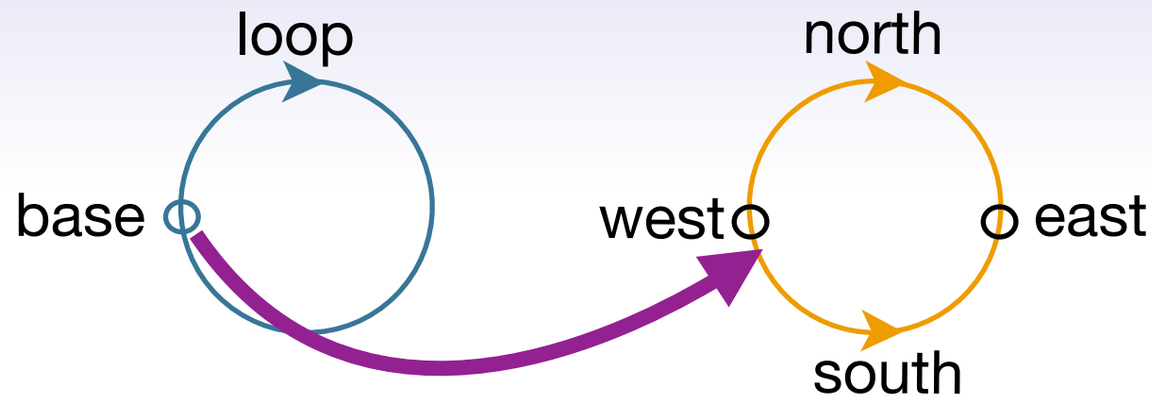




$$\forall x,\ \text{two2one}(\text{one2two}\ x) = x$$

```
Case for base: two2one(one2two base)
                = two2one(west)
                = base
```

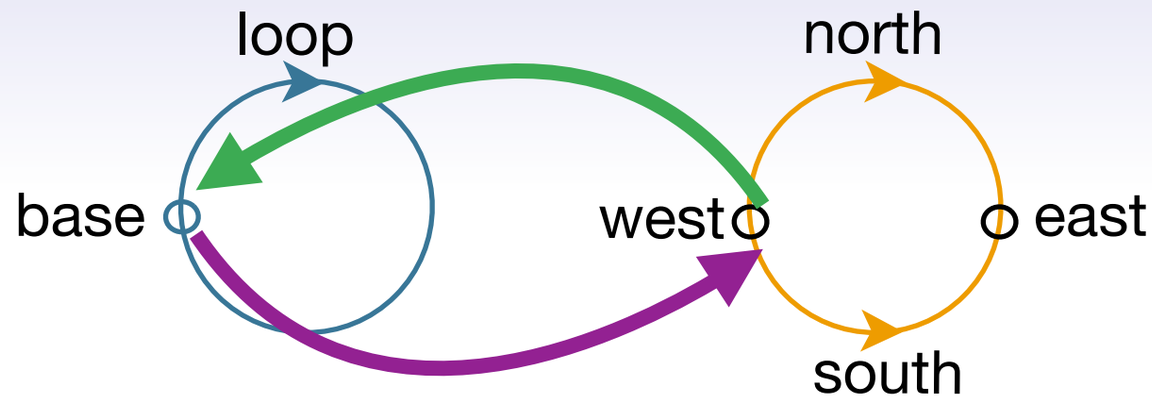
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```



$$\forall x,\ \mathrm{two2one}(\mathrm{one2two}\ x) = x$$

```
Case for base: two2one(one2two base)
                = two2one(west)
                = base
```

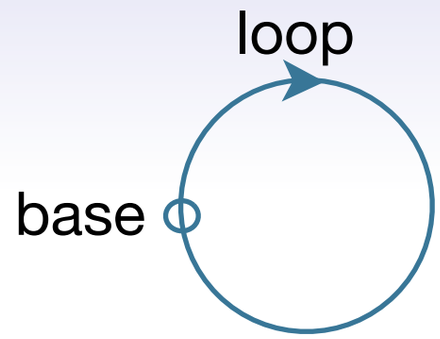
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

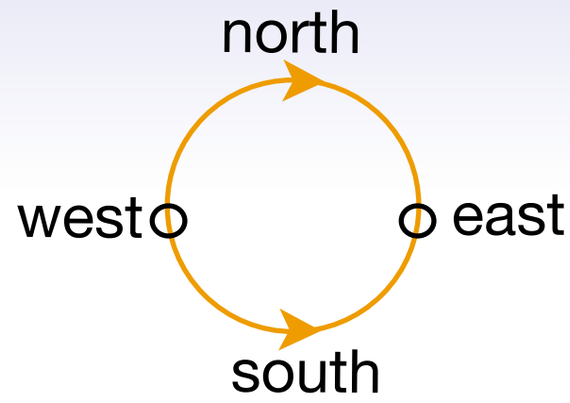


$$\forall x,\ \mathrm{two2one}(\mathrm{one2two}\ x) = x$$

```
Case for base: two2one(one2two base)
                = two2one(west)
                = base
```

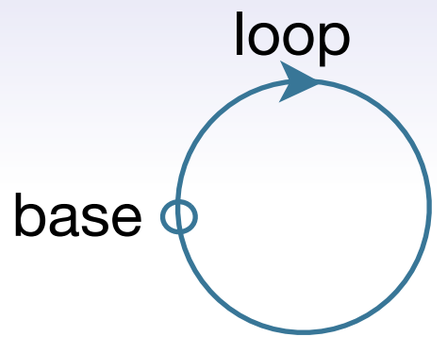
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

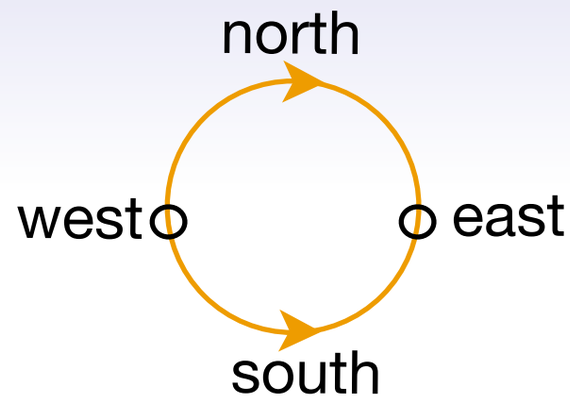




$$\forall x,\ \mathtt{two2one}(\mathtt{one2two}\ x) = x$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```



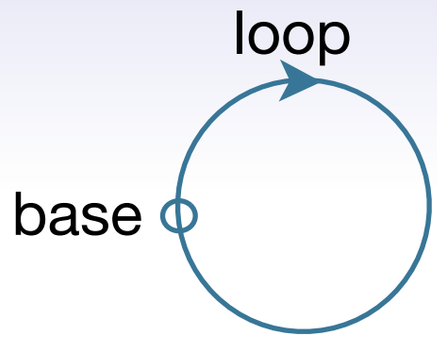


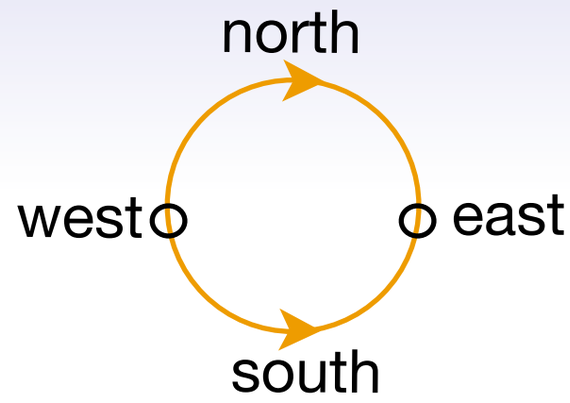
$$\forall x,\ \mathrm{two2one}(\mathrm{one2two}\ x) = x$$

Case for loop:

$$\begin{aligned}
&\mathrm{two2one}(\mathrm{one2two}\ \mathrm{loop}) \\
=\ &\mathrm{two2one}(\mathrm{south}^{-1} \circ \mathrm{north}) \\
=\ &\mathrm{two2one}(\mathrm{south}^{-1}) \circ \mathrm{two2one}(\mathrm{north}) \\
=\ &\mathrm{two2one}(\mathrm{south})^{-1} \circ \mathrm{two2one}(\mathrm{north}) \\
=\ &\mathrm{id}^{-1} \circ \mathrm{loop} \\
=\ &\mathrm{id} \circ \mathrm{loop} \\
=\ &\mathrm{loop}
\end{aligned}$$

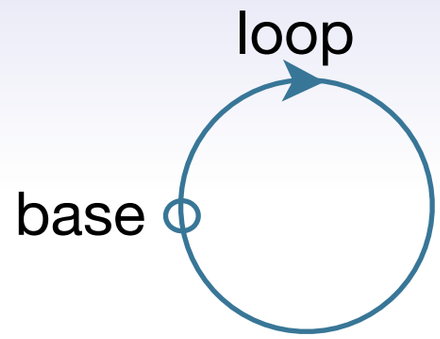
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

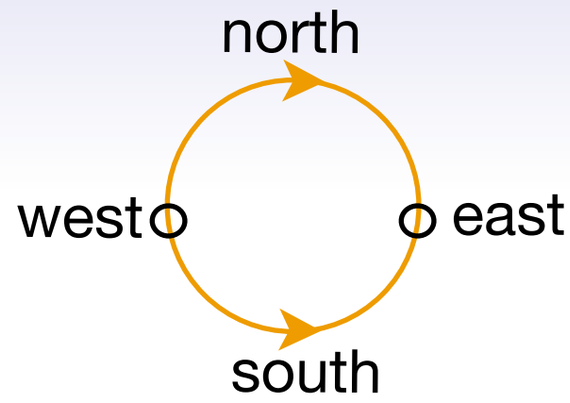




$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```



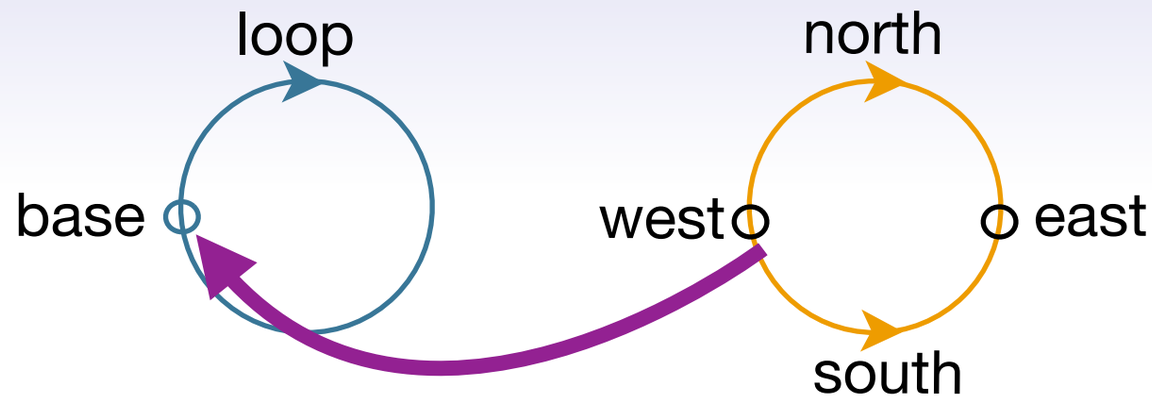


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for west:

```
   one2two(two2one west)
= one2two(base)
= west
```

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

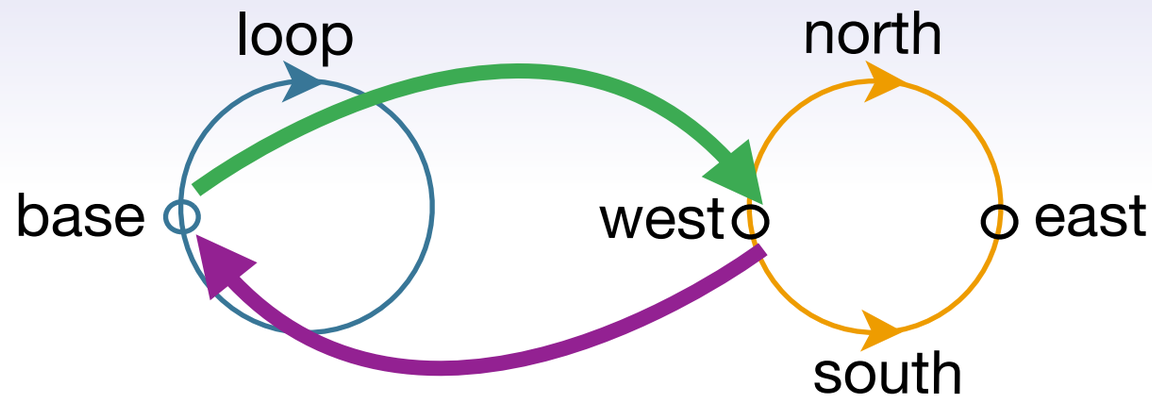


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for west:

$$\begin{aligned} &\mathrm{one2two}(\mathrm{two2one}\ \mathrm{west}) \\ =\ &\mathrm{one2two}(\mathrm{base}) \\ =\ &\mathrm{west} \end{aligned}$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

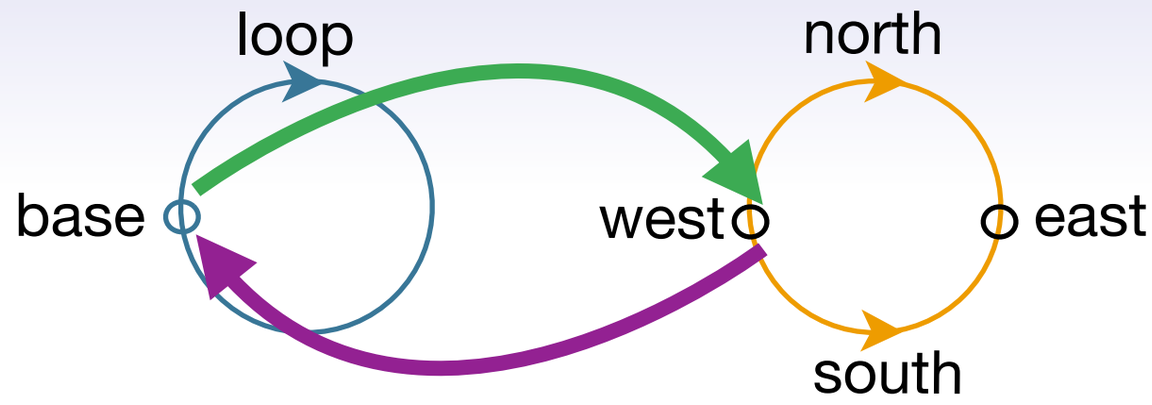


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for west:

$$\begin{aligned} & \mathrm{one2two}(\mathrm{two2one}\ \mathrm{west}) \\ = \ & \mathrm{one2two}(\mathrm{base}) \\ = \ & \mathrm{west} \end{aligned}$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```



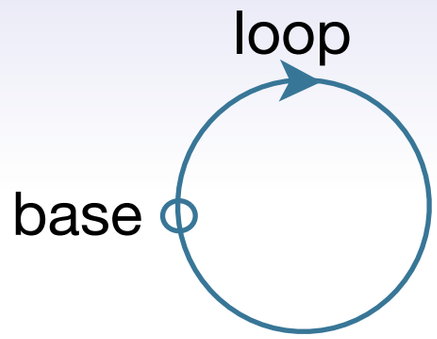
$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

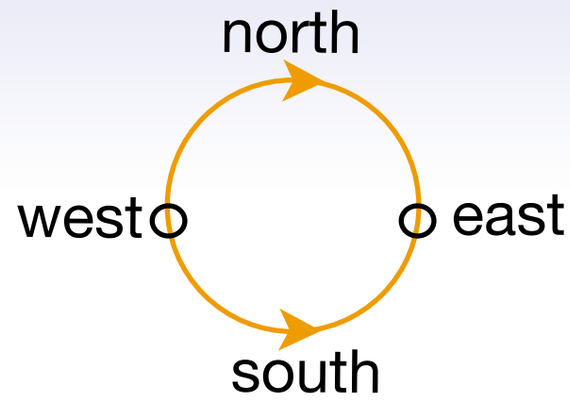
Case for west:

```
   one2two(two2one west)
= one2two(base)
= west
```

use id

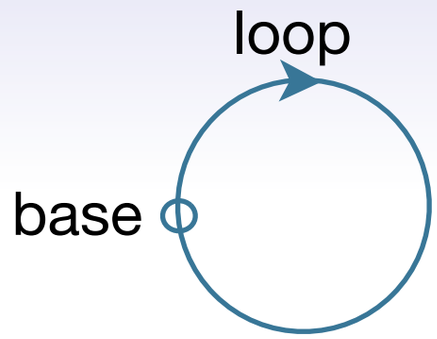
```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

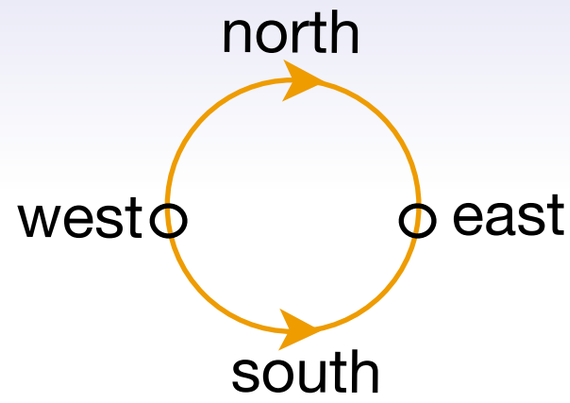




$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```



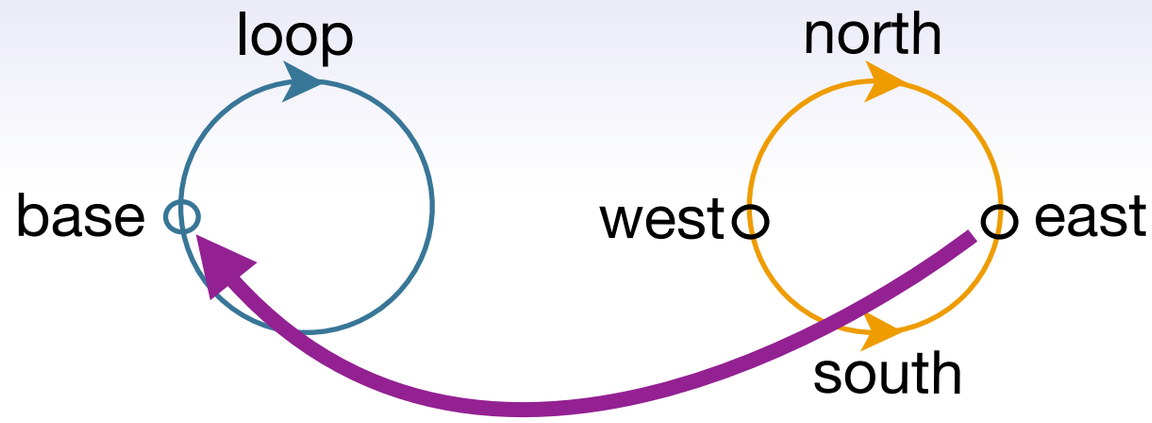


$$\forall x,\ \text{one2two}(\text{two2one}\ x) = x$$

Case for east:

```
  one2two(two2one east)
= one2two(base)
= west
= east
```

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```

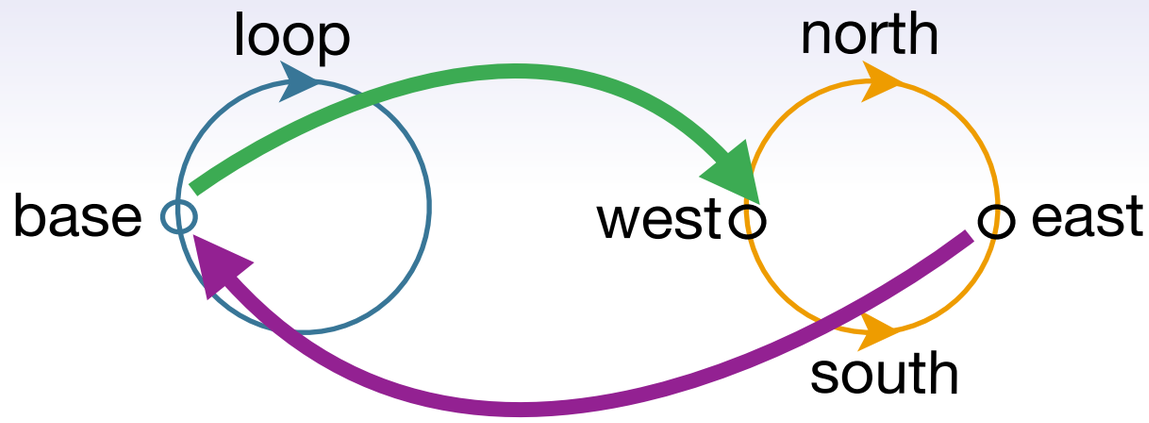


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for east:

$$\begin{aligned} & \mathrm{one2two}(\mathrm{two2one}\ \mathrm{east}) \\ = {} & \mathrm{one2two}(\mathrm{base}) \\ = {} & \mathrm{west} \\ = {} & \mathrm{east} \end{aligned}$$

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```

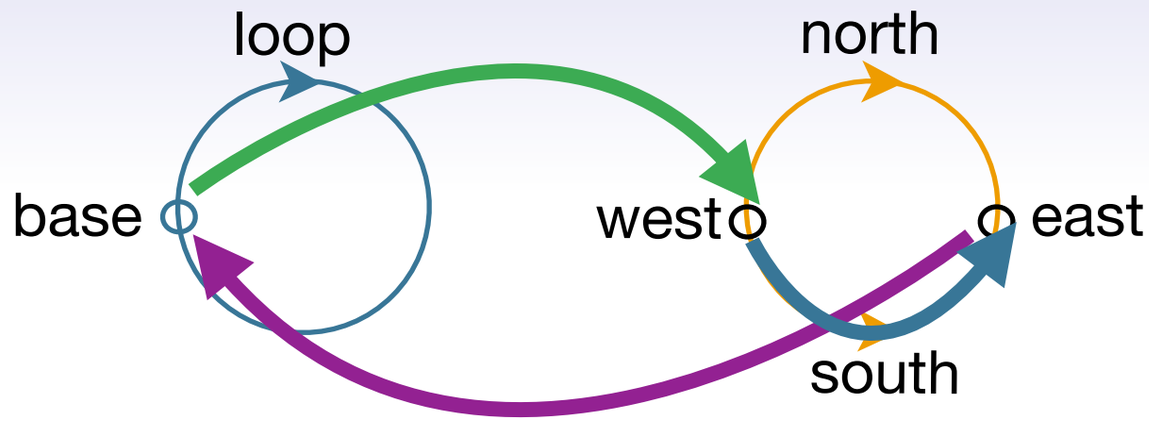


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for east:

```
   one2two(two2one east)
= one2two(base)
= west
= east
```

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```

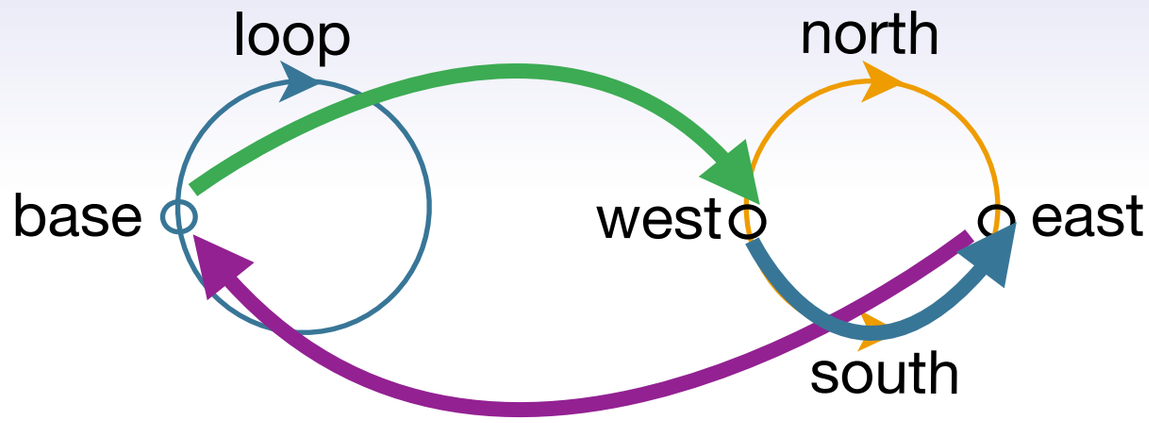


$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

Case for east:

```
   one2two(two2one east)
= one2two(base)
= west
= east
```

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```



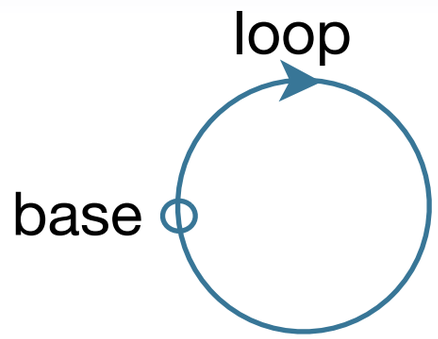
$$\forall x,\ \mathrm{one2two}(\mathrm{two2one}\ x) = x$$

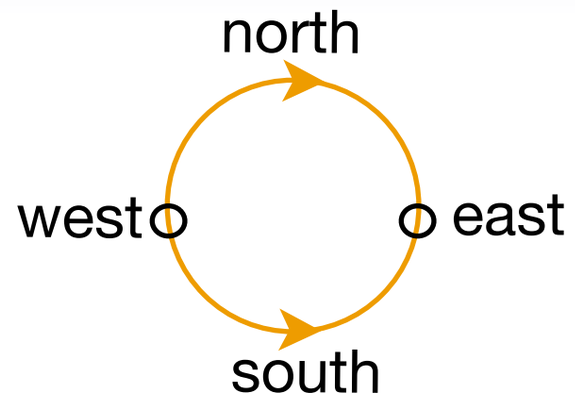
Case for east:

```
   one2two(two2one east)
= one2two(base)                    use south
= west
= east
```

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```



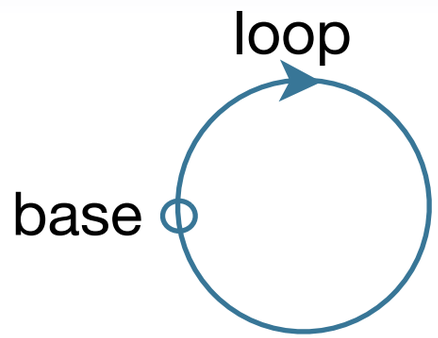


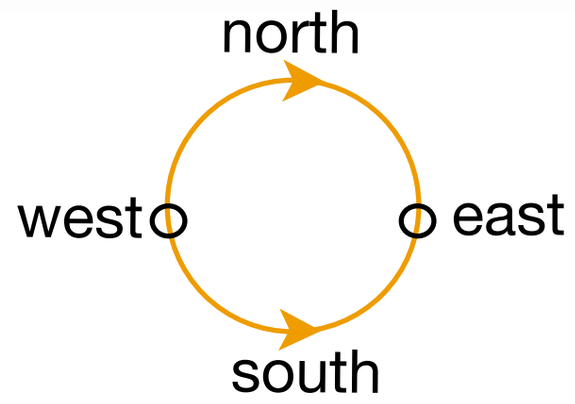

```
          id : one2two(two2one west) = west
       south : one2two(two2one east) = east
```

Case for north:

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id
```






```
one2two base = west
one2two loop = south⁻¹ o north
```

```
id : one2two(two2one west) = west
south : one2two(two2one east) = east
```

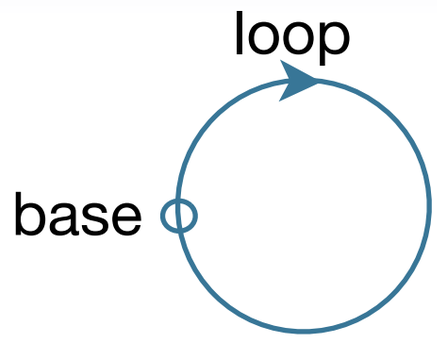
Case for north:

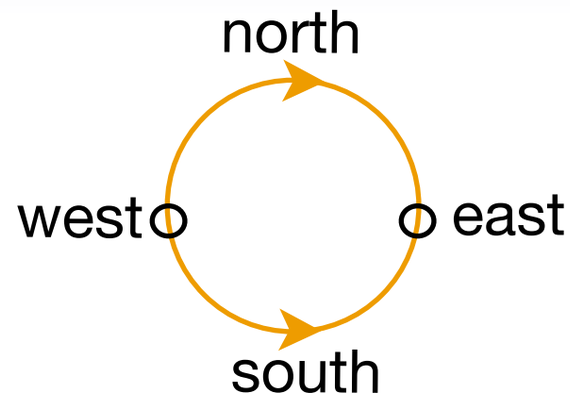
one2two(two2one north)

north

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ ∘ north
```





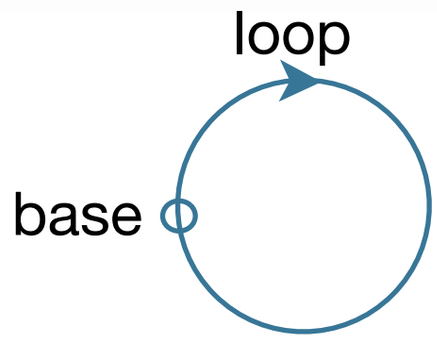

```
                id : one2two(two2one west) = west
             south : one2two(two2one east) = east
```

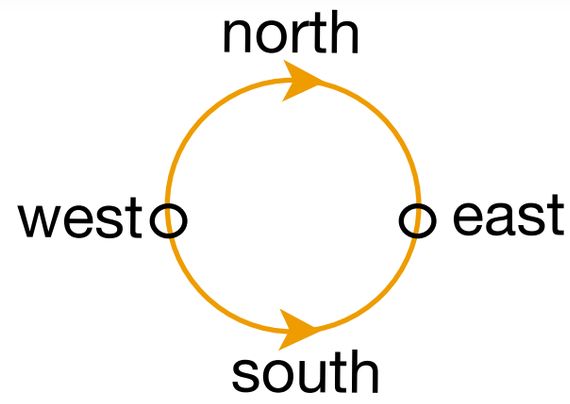
Case for north:

```
    south ∘ one2two(two2one north) = north
```

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id

one2two base = west
one2two loop = south⁻¹ o north
```





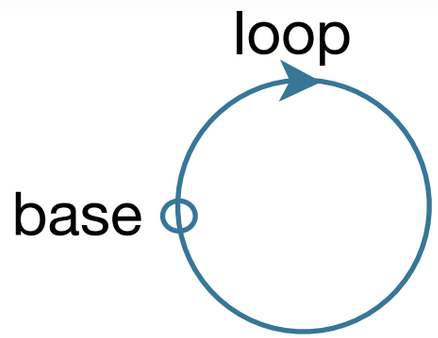

```
                id : one2two(two2one west) = west
             south : one2two(two2one east) = east
```

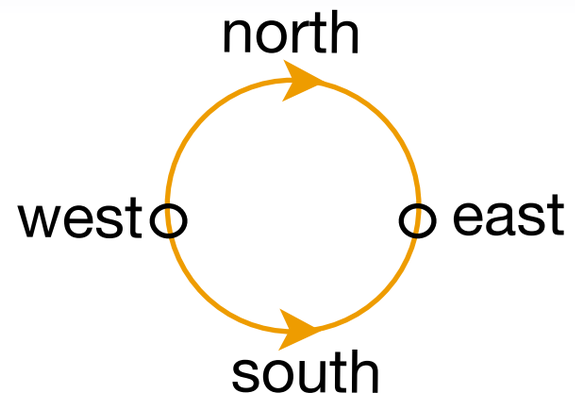
Case for north:

```
    south o one2two(two2one north) = north
    = south o one2two(loop)
    = south o south⁻¹ o north
    = north
```

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id
```






```
one2two base = west
one2two loop = south⁻¹ o north
```

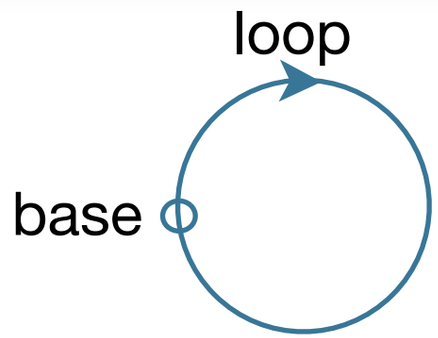
```
      id : one2two(two2one west) = west
south : one2two(two2one east) = east
```

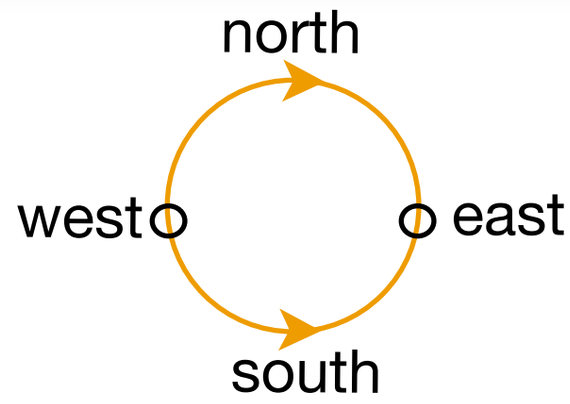
Case for south:

one2two(two2one south)

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id
```






```
one2two base = west
one2two loop = south⁻¹ o north
```

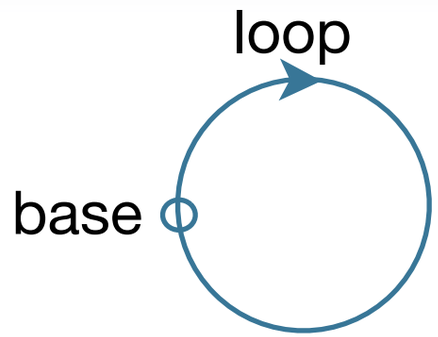
```
id : one2two(two2one west) = west
south : one2two(two2one east) = east
```

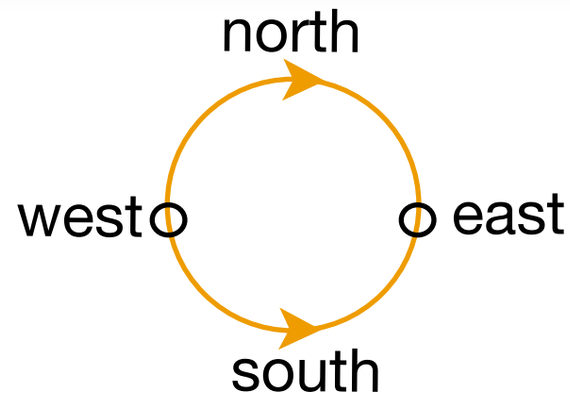
Case for south:

```
south o one2two(two2one south) = south
```

∀x, one2two(two2one x) = x

```
two2one west = base
two2one east = base
two2one north = loop
two2one south = id
```






```
one2two base = west
one2two loop = south⁻¹ o north
```

```
id : one2two(two2one west) = west
south : one2two(two2one east) = east
```

Case for south:

```
south o one2two(two2one south) = south
= south o one2two(id)
= south o id
= south
```

```
one2two : Circle.S¹ -> Circle2.S¹
one2two = Circle.S¹-rec Circle2.w ((! Circle2.s) • Circle2.n)

two2one : Circle2.S¹ → Circle.S¹
two2one = Circle2.S¹-rec Circle.base Circle.base Circle.loop id

comp1 : (x : Circle.S¹) → two2one (one2two x) == x
comp1 = Circle.S¹-elimo _ id
        (PathOver=.in-PathOver-= (vertical-degen-square
          (ap (λ z → two2one (one2two z)) Circle.loop     =⟨ ap-o two2one one2two Circle.loop ⟩
           ap two2one (ap one2two Circle.loop)            =⟨ ap (ap two2one) (Circle.βloop/rec _ _) ⟩
           ap two2one ((! Circle2.s) • Circle2.n)         =⟨ ap-• two2one (! Circle2.s) (Circle2.n) ⟩
           ap two2one (! Circle2.s) • ap two2one Circle2.n =⟨ ap (λ h → h • ap two2one Circle2.n) (ap-! two2one Circle2.s) ⟩
           ! (ap two2one Circle2.s) • ap two2one Circle2.n =⟨ ap (λ h → ! (ap two2one Circle2.s) • h) (Circle2.βn/rec Circle.base Circle.base Circle.loop id) ⟩
           ! (ap two2one Circle2.s) • Circle.loop         =⟨ ap (λ h → ! h • Circle.loop) (Circle2.βs/rec Circle.base Circle.base Circle.loop id) ⟩
           ! id • Circle.loop                             =⟨ •-unit-l Circle.loop ⟩
           Circle.loop                                    =⟨ ! (ap-id Circle.loop) ⟩
           ap (λ z → z) Circle.loop ∎)))

comp2 : (x : Circle2.S¹) → one2two (two2one x) == x
comp2 = Circle2.S¹-elim _
        id
        Circle2.s
        (PathOver=.in-PathOver-= (disc-to-square (!
          (Circle2.s • ap (λ z → one2two (two2one z)) Circle2.n  =⟨ ap (λ x → Circle2.s • x) (ap-o one2two two2one Circle2.n) ⟩
           Circle2.s • ap one2two (ap two2one Circle2.n)        =⟨ ap (λ x → Circle2.s • ap one2two x) (Circle2.βn/rec Circle.base Circle.base Circle.loop id) ⟩
           Circle2.s • ap one2two (Circle.loop)                 =⟨ ap (λ h → Circle2.s • h) (Circle.βloop/rec _ _) ⟩
           Circle2.s • (! Circle2.s • Circle2.n)                =⟨ •-assoc Circle2.s (! Circle2.s) Circle2.n ⟩
           (Circle2.s • ! Circle2.s) • Circle2.n                =⟨ ap (λ h → h • Circle2.n) (!-inv-r Circle2.s) ⟩
           (id) • Circle2.n                                     =⟨ •-unit-l Circle2.n ⟩
           Circle2.n                                            =⟨ ! (ap-id Circle2.n) ⟩
           ap (λ z → z) Circle2.n ∎))))
        (PathOver=.in-PathOver-= (disc-to-square (!
          (Circle2.s • ap (λ z → one2two (two2one z)) Circle2.s  =⟨ ap (λ x → Circle2.s • x) (ap-o one2two two2one Circle2.s) ⟩
           Circle2.s • ap one2two (ap two2one Circle2.s)        =⟨ ap (λ x → Circle2.s • ap one2two x) (Circle2.βs/rec Circle.base Circle.base Circle.loop id) ⟩
           Circle2.s                                            =⟨ ! (ap-id Circle2.s) ⟩
           ap (λ z → z) Circle2.s ∎))))
```

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_2$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Blakers-Massey

$T^2 = S^1 \times S^1$

Van Kampen

Covering spaces

Whitehead for n-types

Cohomology axioms

Mayer-Vietoris

[Brunerie, Cavallo, Finster, Hou, Licata, Lumsdaine, Shulman]

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_2$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Blakers-Massey

$T^2 = S^1 \times S^1$

Van Kampen

Covering spaces

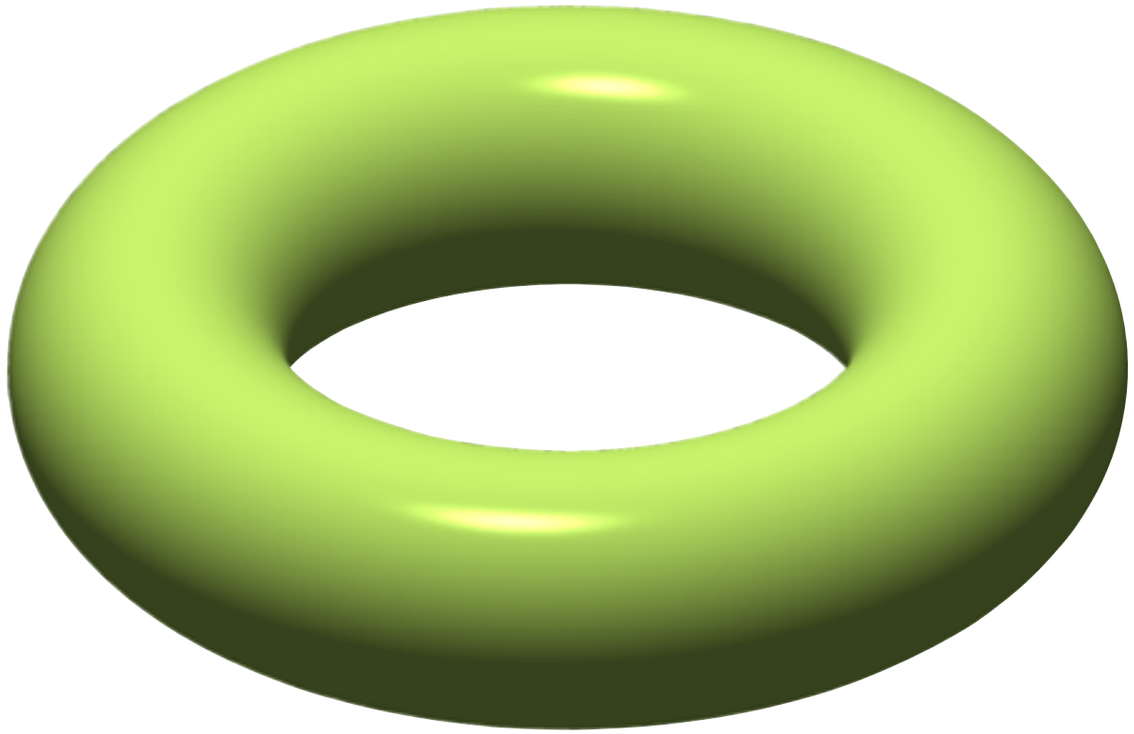
Whitehead for n-types

Cohomology axioms

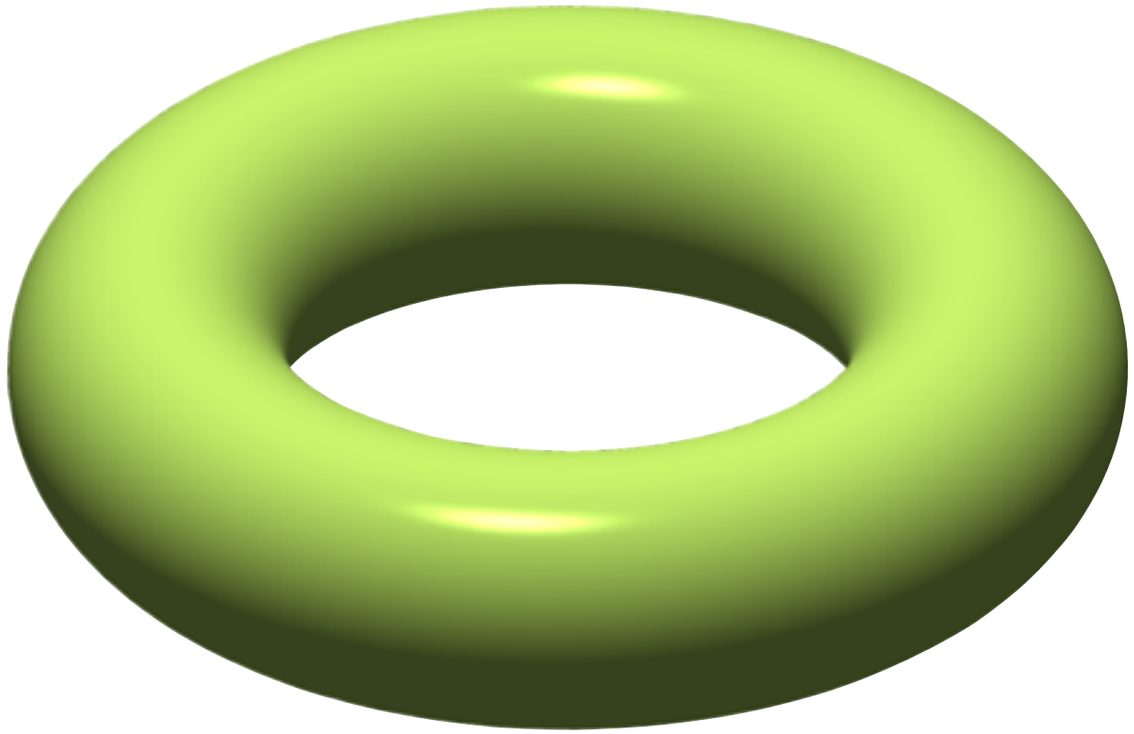
Mayer-Vietoris

[Brunerie, Cavallo, Finster, Hou, Licata, Lumsdaine, Shulman]

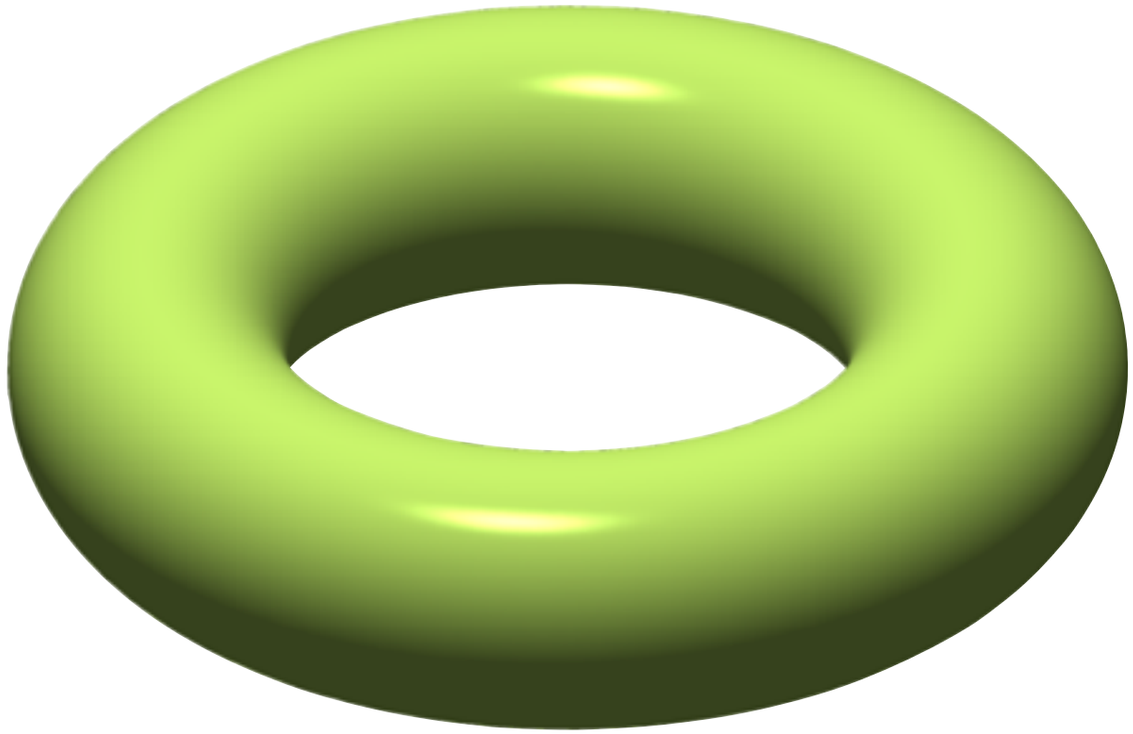
Torus

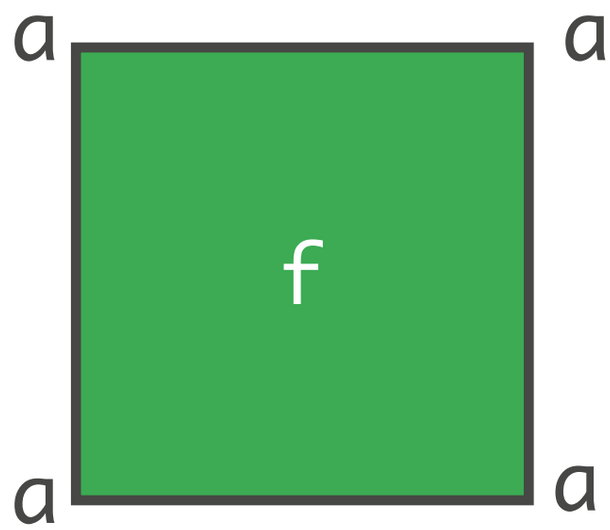


Torus

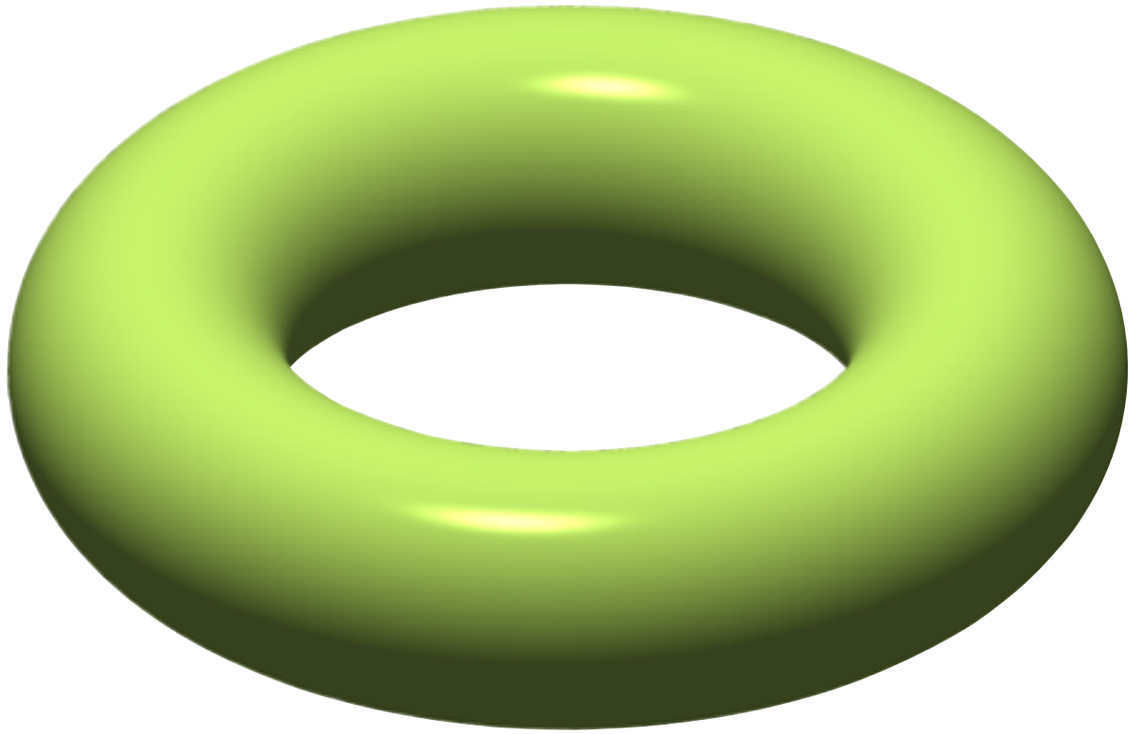


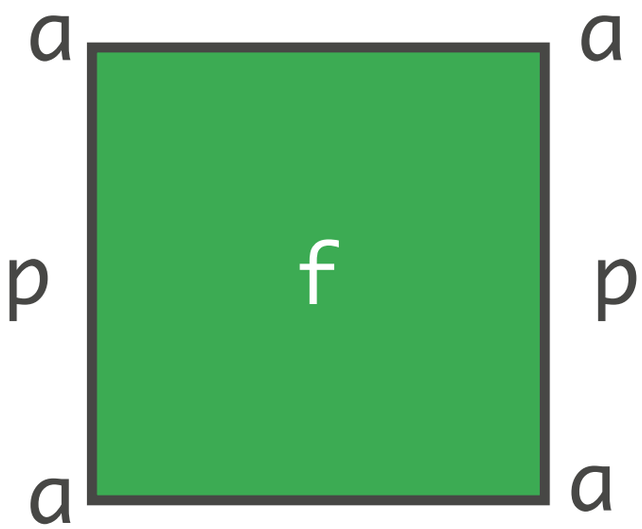
Torus



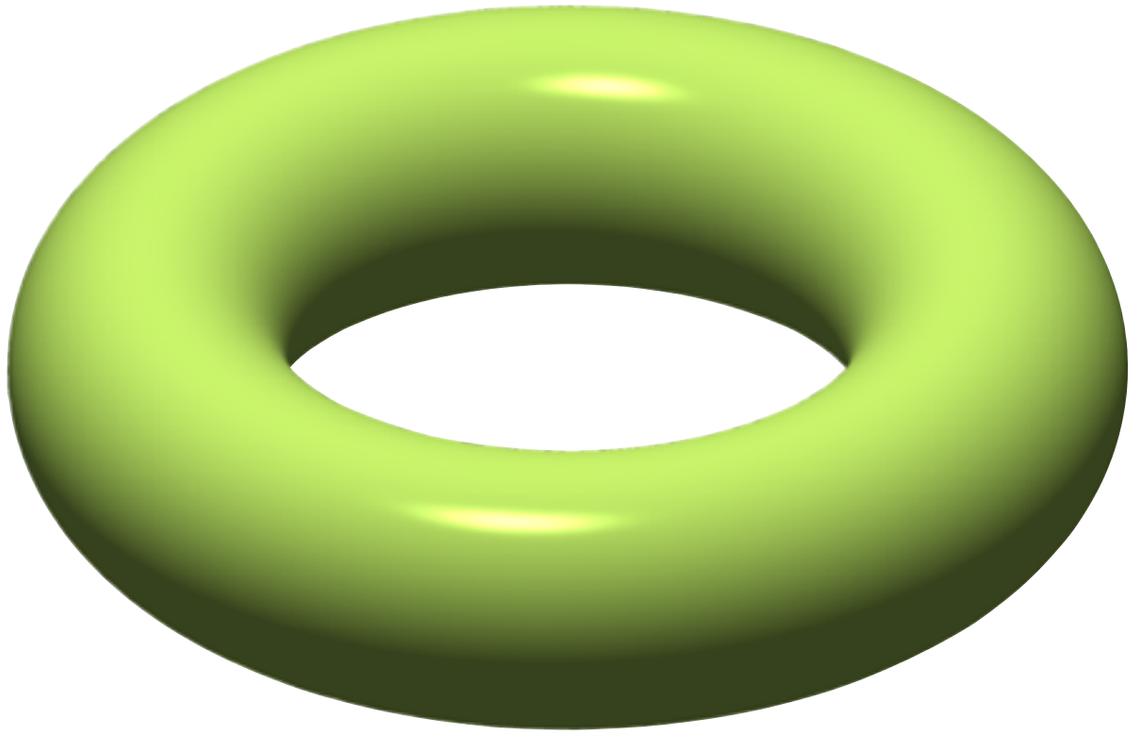


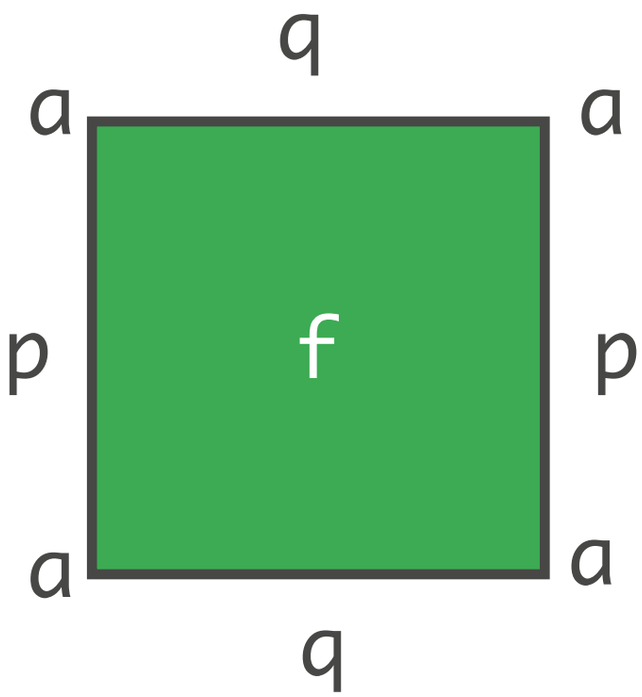
Torus



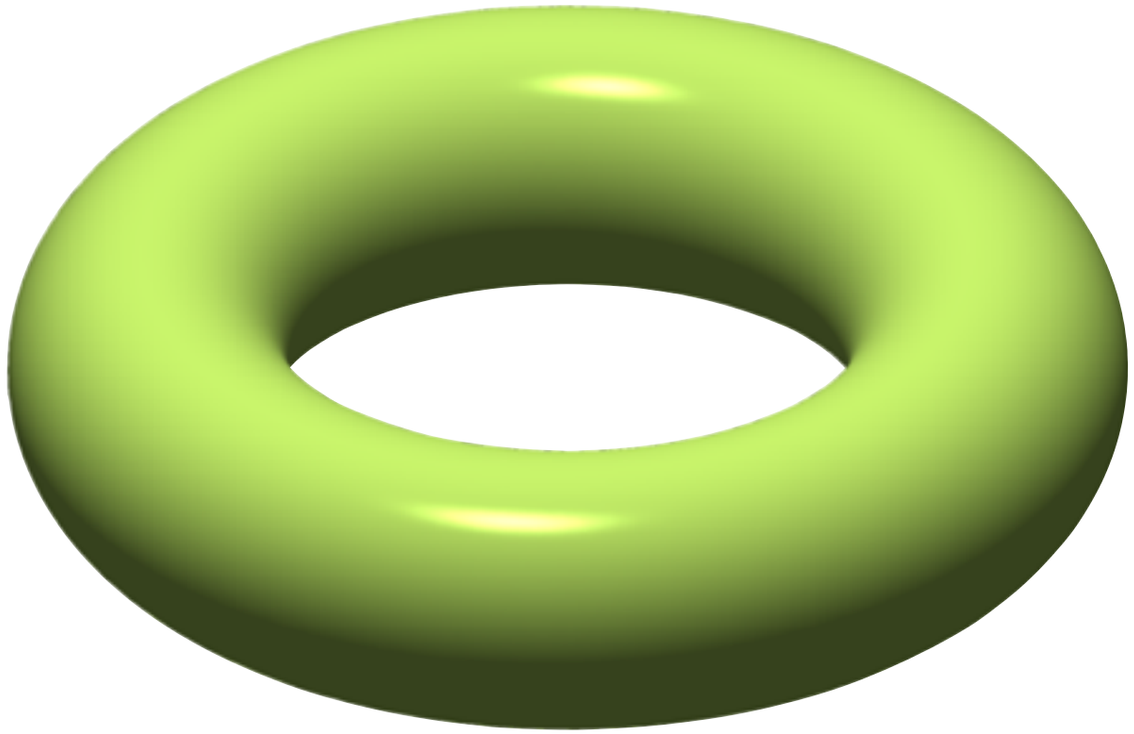


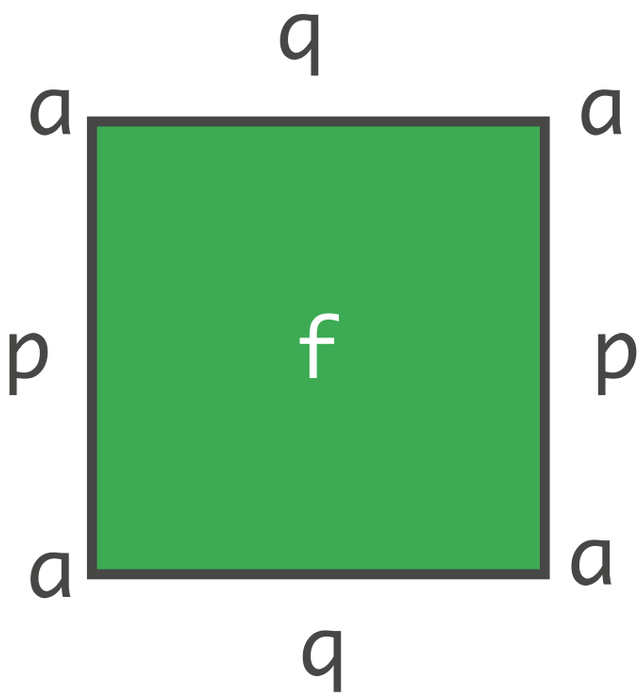
Torus





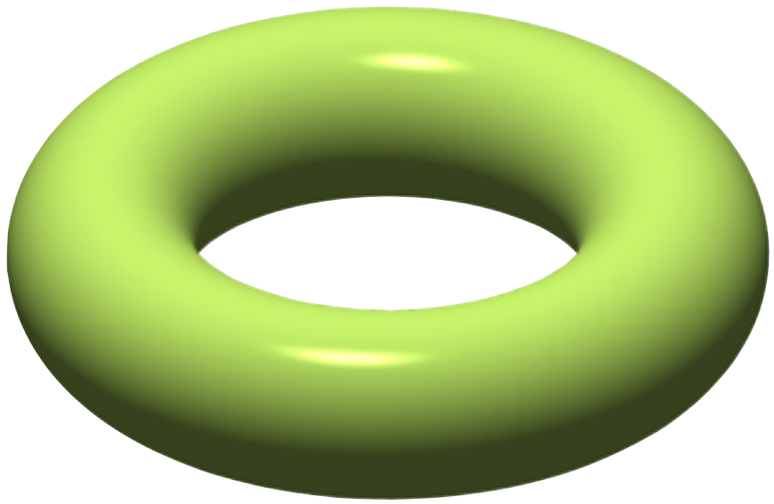
Torus

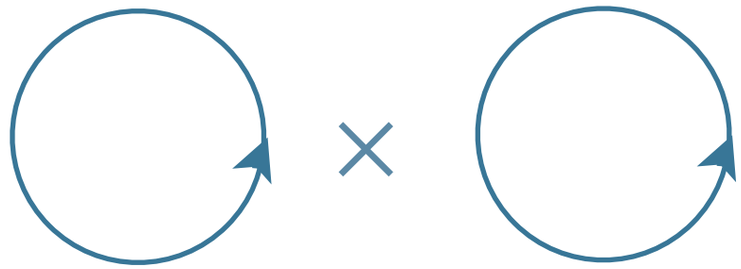




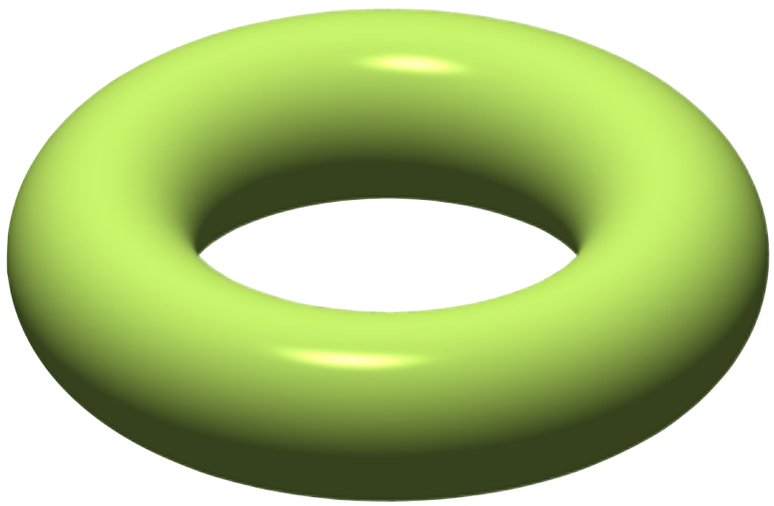

```
a   : Torus
p,q : a = a
f   : q o p = p o q
```

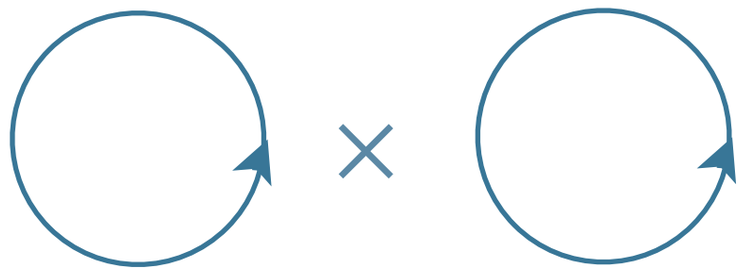
Torus = Circle × Circle





Torus = Circle × Circle





```
t2c : Torus → Circle × Circle
t2c a = (base,base)
t2c p = (loop,id)
t2c q = (id,loop)
t2c f = … : (loop,id)o(id,loop)
          = (id,loop)o(loop,id)
```

$$\frac{\alpha : x =_A x' \qquad \beta : y =_B y'}{(\alpha,\beta) : (x,y) =_{A\times B} (x',y')}$$

$$(\alpha_2,\beta_2) \circ (\alpha_1,\beta_1) = (\alpha_2 \circ \alpha_1, \beta_2 \circ \beta_1)$$

```
t2c : Torus → Circle × Circle
t2c a = (base,base)
t2c p = (loop,id)
t2c q = (id,loop)
t2c f = … : (loop,id)o(id,loop)
          = (id,loop)o(loop,id)
```

$$\frac{\alpha : x =_A x' \qquad \beta : y =_B y'}{(\alpha,\beta) : (x,y) =_{A\times B} (x',y')}$$

$$(\alpha_2,\beta_2) \circ (\alpha_1,\beta_1) = (\alpha_2 \circ \alpha_1, \beta_2 \circ \beta_1)$$

```
t2c : Torus → Circle × Circle
t2c a = (base,base)
t2c p = (loop,id)
t2c q = (id,loop)
t2c f = … : (loop,id)o(id,loop)
          = (id,loop)o(loop,id)
```

both equal (loop,loop)

$$\frac{\alpha : x =_A x' \qquad \beta : y =_B y'}{(\alpha,\beta) : (x,y) =_{A\times B} (x',y')}$$

$$(\alpha_2,\beta_2) \circ (\alpha_1,\beta_1) = (\alpha_2 \circ \alpha_1, \beta_2 \circ \beta_1)$$

```
t2c : T -> S¹ x S¹
t2c = T-rec (S¹.base , S¹.base) (pairx≃ id S¹.loop) (pairx≃ S¹.loop id) (pair-square vrefl-square hrefl-square)

abstract
  c2t-square-and-cube : Σ \ s -> Cube s (square-symmetry T.f)
                                        hrefl-square (horiz-degen-square (S¹.βloop/rec T.a T.p))
                                        (horiz-degen-square (S¹.βloop/rec T.a T.p)) hrefl-square
  c2t-square-and-cube = (fill-cube-left (square-symmetry T.f)
                          hrefl-square
                          (horiz-degen-square (S¹.βloop/rec T.a T.p)) (horiz-degen-square (S¹.βloop/rec T.a T.p)) hrefl-square)

c2t-square : Square T.q (ap (λ z → S¹-rec T.a T.p z) S¹.loop) (ap (λ z → S¹-rec T.a T.p z) S¹.loop) T.q
c2t-square = fst c2t-square-and-cube

c2t-loop-homotopy = (S¹-elimo (\ x -> (S¹-rec T.a T.p) x == (S¹-rec T.a T.p) x) T.q (PathOver=.in-PathOver-= c2t-square))

c2t' : S¹ → S¹ → T
c2t' x y = S¹-rec (S¹-rec T.a T.p) (λ≃ c2t-loop-homotopy) x y

c2t : S¹ x S¹ → T
c2t (x , y) = c2t' x y
```

```
cube5 : Σ \ square1'' → Σ \ square2'' →
      Cube (bifunctor-square1 c2t' S¹.loop S¹.loop) (square-symmetry T.f) square2'' square1'' square1'' square2''
cube5 = _ , _ , (
        SquareOver=ND.out-SquareOver-= (apdo-by-equals _ _ S¹.loop (λ≃ reduce-c2t')) ∙-cube-h
        degen-cube-h (ap PathOver=.out-PathOver-= (S¹.βloop/elimo _ T.q (PathOver=.in-PathOver-= c2t-square))) ∙-cube-h
        degen-cube-h (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) _)
          ∙-cube-h (snd c2t-square-and-cube))

t2c2t : (x : T) → c2t (t2c x) == x
t2c2t = T-elim (\ x -> c2t (t2c x) == x)
              id
              (PathOver=.in-PathOver-= (square-symmetry square1))
              (PathOver=.in-PathOver-= (square-symmetry square2))
              (SquareOver=ND.in-SquareOver-=
                (whisker-cube (! (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) (square-symmetry square1)))
                              (! (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) (square-symmetry square1)))
                              (! (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) (square-symmetry square2)))
                              id id
                              (! (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) (square-symmetry square2)))
                              (cube-symmetry-left-to-top goal1))) where
      square1 = _
      square2 = _
      goal1 : Cube (ap-square (λ z → c2t (t2c z)) T.f)
                   (ap-square (λ z → z) T.f)
                   square1 square2 square2 square1
      goal1 = (ap-square-o c2t t2c T.f) ∙-cube-h
              ap-cube c2t
                (T.βf/rec (S¹.base , S¹.base) (pairx≃ id S¹.loop)
                  (pairx≃ S¹.loop id) (pair-square vrefl-square hrefl-square)) ∙-cube-h
              bifunctor-cube1 c2t' S¹.loop S¹.loop ∙-cube-h
              cube-square-symmetry-left (snd (snd cube5)) ∙-cube-h
              degen-cube-h (square-symmetry-symmetry T.f)  ∙-cube-h
              ap-square-id! T.f
```

```
c2t2c : (x y : S¹) → t2c (c2t' x y) == (x , y)
c2t2c = S¹-elimo _ (S¹-elimo _ id (PathOver=.in-PathOver-= square1))
        (coe (! PathOver=-NDdomain) (\ x -> PathOver=.in-PathOver-=
          (S¹-elimo
            (λ x₁ → Square (S¹-elimo (λ x₂ → t2c (c2t' S¹.base x₂) == (S¹.base , x₂)) id (PathOver=.in-PathOver-= square1) x₁)
                           (ap (λ z → t2c (c2t' z x₁)) S¹.loop)
                           (ap (λ z → z , x₁) S¹.loop)
                           (S¹-elimo (λ x₂ → t2c (c2t' S¹.base x₂) == (S¹.base , x₂)) id (PathOver=.in-PathOver-= square1) x₁))
            square2
            (coe (! (PathOver-square/= S¹.loop square2 square2))
              (transport (λ x₁ → Cube square2 square2
                                      x₁ (PathOver=.out-PathOver-= (apdo (λ x₂ → ap (λ z → t2c (c2t' z x₂)) S¹.loop) S¹.loop))
                                      (PathOver=.out-PathOver-= (apdo (λ x₂ → ap (λ z → z , x₂) S¹.loop) S¹.loop)) x₁)
                  (! (IsEquiv.β (snd PathOver=.out-PathOver-=-eqv) square1 ∘
                     ap PathOver=.out-PathOver-= (S¹.βloop/elimo _ id (PathOver=.in-PathOver-= square1))))
                  cube3)))
          x))) where
  square1' = _
  square2' = _
  cube4 : Cube
          (PathOver=.out-PathOver-=
            (apdo (λ x₁ → ap (λ z → t2c (c2t' z x₁)) S¹.loop) S¹.loop))
          (PathOver=.out-PathOver-=
            (apdo (λ x₁ → ap (λ z → z , x₁) S¹.loop) S¹.loop))
          square1' square2' square2' square1'
  cube4 = !-cube-h (bifunctor-cube1' (λ x y → t2c (c2t' x y)) S¹.loop S¹.loop) ∙-cube-h
          ap-square-o t2c c2t (pair-square hrefl-square vrefl-square) ∙-cube-h
          ap-cube t2c (bifunctor-cube1' c2t' S¹.loop S¹.loop ∙-cube-h (snd (snd cube5))) ∙-cube-h
          degen-cube-h (ap-square-symmetry t2c T.f) ∙-cube-h
          cube-square-symmetry-left (T.βf/rec (S¹.base , S¹.base) (pairx≃ id S¹.loop) (pairx≃ S¹.loop id) (pair-square vrefl-square hrefl-square)) ∙-cube-h
          degen-cube-h (pair-vrefl-hrefl-symmetry S¹.loop S¹.loop) ∙-cube-h
          (ap-square-id! (pair-square hrefl-square vrefl-square)) ∙-cube-h
          (bifunctor-cube1' _,_ S¹.loop S¹.loop)

  square1 = _
  square2 = _
  cube3 : Cube square2 square2 square1
          (PathOver=.out-PathOver-= (apdo (λ x₁ → ap (λ z → t2c (c2t' z x₁)) S¹.loop) S¹.loop))
          (PathOver=.out-PathOver-= (apdo (λ x₁ → ap (λ z → z , x₁) S¹.loop) S¹.loop))
          square1
  cube3 = (cube-symmetry-left-to-top cube4)
```

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_2$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Blakers-Massey

$T^2 = S^1 \times S^1$

Van Kampen

Covering spaces

Whitehead for n-types

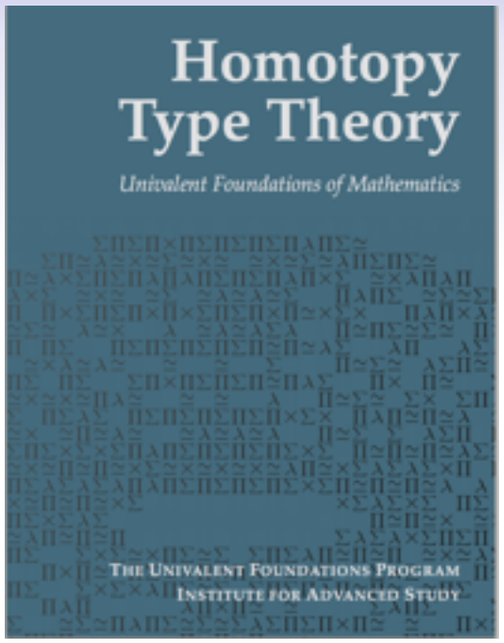
Cohomology axioms

Mayer-Vietoris

[Brunerie, Cavallo, Finster, Hou, Licata, Lumsdaine, Shulman]

Conclusion

Papers and code



1. The HoTT Book, `homotopytypetheory.org`

2. Homotopy theory in Agda:
   - Fundamental group of the circle [LICS'13]
   - $\pi_n(S^n) = \mathbb{Z}$ [CPP'13]
   - Eilenberg-MacLane spaces [LICS'14]
   - `github.com/dlicata335/`
   - `github.com/hott/hott-agda`

3. Computation:
   - 2D Type Theory [POPL'12]
   - Homotopical Patch Theory [ICFP'14]
   - Directed Type Theory [MFPS'11]

Homotopy Type Theory