

# Directed Type Theory for State Space Analysis

Paige Randall North

University of Pennsylvania

21 July 2021

# Overview

- ▶ **(Dependent) type theory** is a foundation for mathematics in which all proofs  $\leftrightarrow^1$  programs can be checked  $\leftrightarrow^1$  compiled by a computer.
- ▶ **Homotopy** type theory is a foundation for the study of **homotopy types** (topological spaces).
- ▶ **Directed** homotopy type theory<sup>2</sup> is a foundation for the study of **directed** homotopy types (**directed** topological spaces).
- ▶ Directed spaces<sup>3</sup> capture much of the theory of vector fields on manifolds<sup>4</sup>

---

<sup>1</sup>Curry-Howard correspondence

<sup>2</sup>under construction

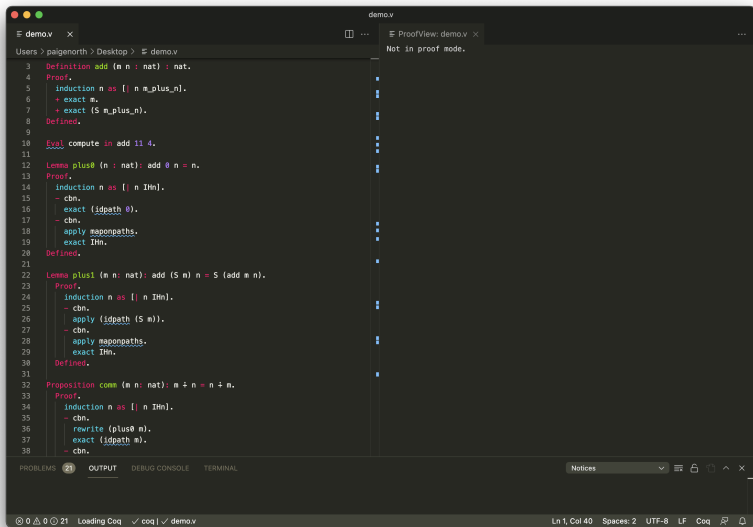
<sup>3</sup>See Sanjeevi Krishnan's talk for more details about directed spaces.

<sup>4</sup>See Jared Culbertson's and Samuel Burden's talks too see how such objects are used to provide operational semantics for robotics.

# Type theory

- ▶ The basic objects are types, that we can interpret as sets, propositions, a program specification, etc.
- ▶ Built out of type formers. We can construct:
  - ▶ types like  $\mathbb{N}$  and
  - ▶ types like  $A \times B$ ,  $A + B$ ,  $A \rightarrow B$ , etc, from two types  $A$  and  $B$ .
- ▶ The type formers are (usually) given by inductive principles.
  - ▶ E.g.:  $\mathbb{N}$  is inductively generated from the *canonical terms*  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for every  $n : \mathbb{N}$ .

# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers. The script defines an addition function, proves its correctness for zero and successors, and states a commutativity proposition. The interface includes a top bar with window controls, a left sidebar with file explorer, a main editor area, and a bottom status bar with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The status bar also shows the current line and column (Ln 1, Col 40), the number of spaces (2), the encoding (UTF-8), and the file type (LF Coq).

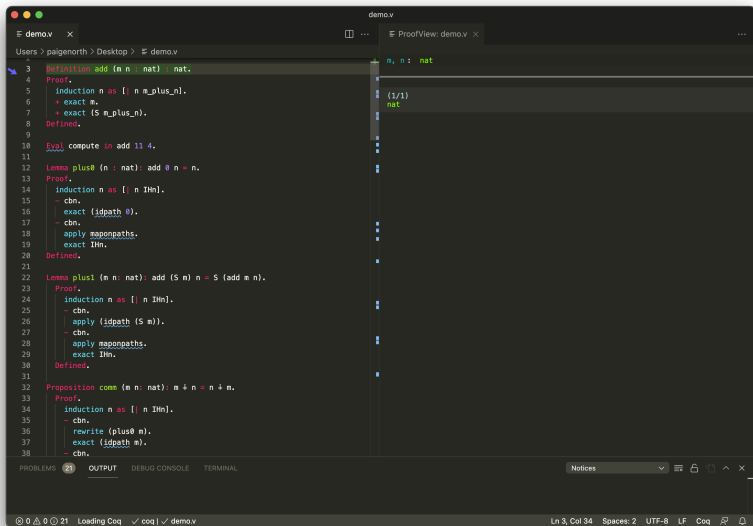
```
demo.v
Users > paigenorth > Desktop > demo.v

3 Definition add (n m : nat) : nat.
4   Proof.
5     induction n as [ | n m_plus_n].
6     + exact m.
7     + exact (S m_plus_n).
8   Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat): add 0 n = n.
13 Proof.
14   induction n as [ | n IHn].
15   - cbn.
16     exact (idpath 0).
17   - cbn.
18     apply maponpaths.
19     exact IHn.
20 Defined.
21
22 Lemma plus1 (m n: nat): add (S m) n = S (add m n).
23 Proof.
24   induction n as [ | n IHn].
25   - cbn.
26     apply (idpath (S m)).
27   - cbn.
28     apply maponpaths.
29     exact IHn.
30 Defined.
31
32 Proposition comm (m n: nat): m + n = n + m.
33 Proof.
34   induction n as [ | n IHn].
35   - cbn.
36     rewrite (plus0 m).
37     exact (idpath m).
38   - cbn.
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL Notices

0 0 0 21 Loading Coq ✓ coq | demo.v Ln 1, Col 40 Spaces: 2 UTF-8 LF Coq

# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers. The script defines an addition function, proves its correctness for zero and successor, and proves the commutativity of addition. The right-hand pane shows the current goal state, which is  $n, n : \text{nat}$  with a goal of  $(1/1) \text{ nat}$ .

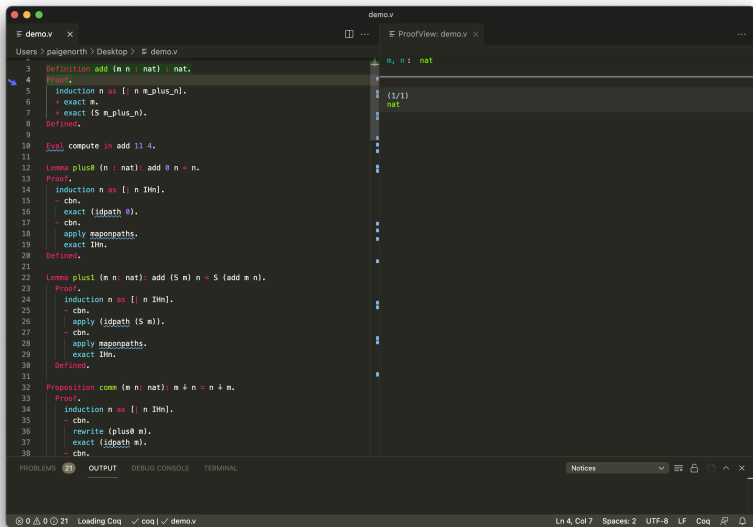
```
3 Definition add (n n' : nat) : nat.
4   Proof.
5     induction n as [ | n m_plus_n].
6     + exact m.
7     + exact (S m_plus_n).
8   Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat) : add 0 n = n.
13 Proof.
14   induction n as [ | n IHn].
15   - cbn.
16     exact (idpath 0).
17   - cbn.
18     apply maponpaths.
19     exact IHn.
20 Defined.
21
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).
23 Proof.
24   induction n as [ | n IHn].
25   - cbn.
26     apply (idpath (S m)).
27   - cbn.
28     apply maponpaths.
29     exact IHn.
30 Defined.
31
32 Proposition comm (m n : nat) : m + n = n + m.
33 Proof.
34   induction n as [ | n IHn].
35   - cbn.
36     rewrite (plus0 m).
37     exact (idpath m).
38   - cbn.
```

Right-hand pane (Goal):

```
n, n : nat
(1/1)
nat
```

Bottom status bar: Loading Coq ✓ coq | demo.v Ln 3, Col 34 Spaces: 2 UTF-8 LF Coq

# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for defining addition on natural numbers. The script includes a `Definition` for `add`, an `Eval` command to compute `add 11 4`, and three lemmas: `plus0`, `plus1`, and `comm`. Each lemma is followed by a `Proof.` block containing `induction` and various tactics like `cbn`, `exact`, `idpath`, `maponpaths`, and `Defined`. The right-hand pane shows the `ProofView` for the current goal, displaying the type `n, n : nat` and the goal `(1/1) nat`. The bottom status bar indicates the Coq version (0.0.0), the current file (`demo.v`), and the current line and column (Ln 4, Col 7).

```
demo.v
Users > paigenorth > Desktop > demo.v

3 Definition add (n n : nat) : nat.
4 Proof.
5   induction n as [ | n m_plus_n].
6   + exact m.
7   + exact (S m_plus_n).
8 Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat) : add 0 n = n.
13 Proof.
14   induction n as [ | n IHn].
15   - cbn.
16     exact (idpath 0).
17   - cbn.
18     apply maponpaths.
19     exact IHn.
20 Defined.
21
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).
23 Proof.
24   induction n as [ | n IHn].
25   - cbn.
26     apply (idpath (S m)).
27   - cbn.
28     apply maponpaths.
29     exact IHn.
30 Defined.
31
32 Proposition comm (m n : nat) : m + n = n + m.
33 Proof.
34   induction n as [ | n IHn].
35   - cbn.
36     rewrite (plus0 m).
37     exact (idpath m).
38   - cbn.
```

# Type theory

The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers, including definitions, lemmas, and a proposition. The right-hand pane shows the proof state for the current goal.

```
3 Definition add (n m : nat) : nat.
4   Proof.
5     induction n as [ | n m_plus_n].
6     + exact m.
7     + exact (S m_plus_n).
8   Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat) : add 0 n = n.
13 Proof.
14   induction n as [ | n IHn].
15   - cbn.
16     exact (idpath 0).
17   - cbn.
18     apply maponpaths.
19     exact IHn.
20 Defined.
21
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).
23 Proof.
24   induction n as [ | n IHn].
25   - cbn.
26     apply (idpath (S m)).
27   - cbn.
28     apply maponpaths.
29     exact IHn.
30 Defined.
31
32 Proposition comm (m n : nat) : m + n = n + m.
33 Proof.
34   induction n as [ | n IHn].
35   - cbn.
36     rewrite (plus0 m).
37     exact (idpath m).
38   - cbn.
```

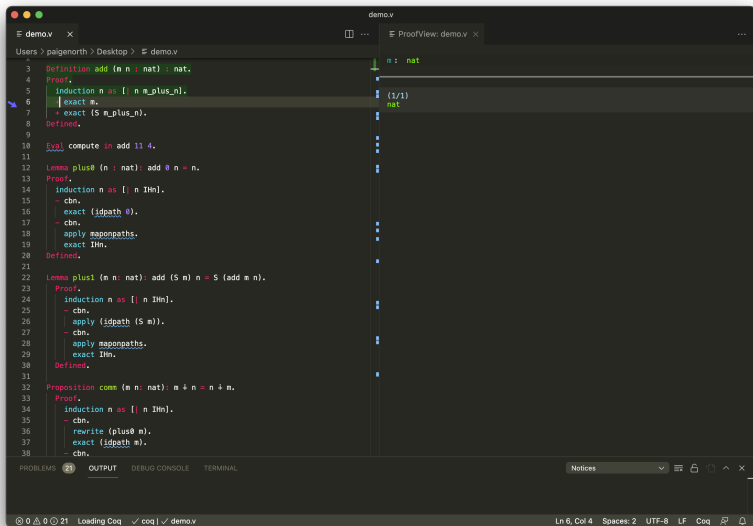
The right-hand pane shows the proof state for the current goal:

```
m : nat
(1/2)
nat
(2/2)
nat
```

The bottom status bar indicates the following information:

- PROBLEMS: 21
- OUTPUT
- DEBUG CONSOLE
- TERMINAL
- Notices
- Ln 5, Col 33
- Spaces: 2
- UTF-8
- LF
- Coq

# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers. The script defines an addition function, proves its correctness for zero and successor, and proves the commutative property. A proof view window on the right shows the current goal.

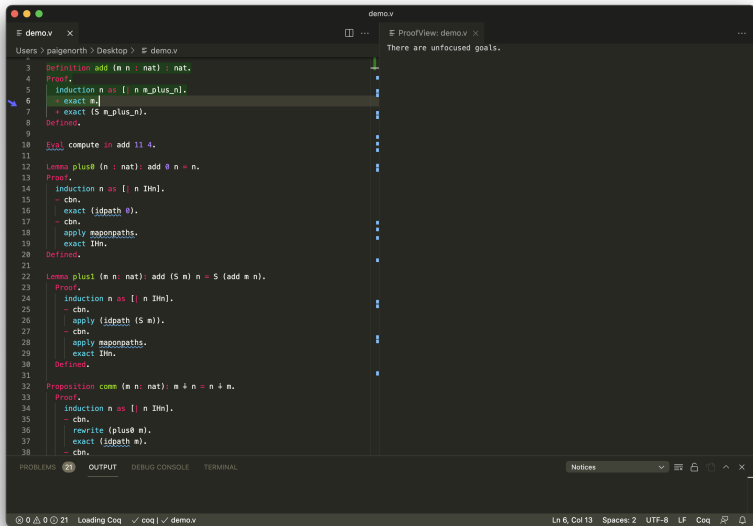
```
1 Definition add (n m : nat) : nat.  
2   Proof.  
3     induction n as [ | n m_plus_n].  
4     - exact m.  
5     - exact (S m_plus_n).  
6   Defined.  
7  
8   Eval compute in add 11 4.  
9  
10  
11  
12 Lemma plus0 (n : nat) : add 0 n = n.  
13 Proof.  
14   induction n as [ | n IHn].  
15   - cbn.  
16     exact (idpath 0).  
17   - cbn.  
18     apply maponpaths.  
19     exact IHn.  
20   Defined.  
21  
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).  
23 Proof.  
24   induction n as [ | n IHn].  
25   - cbn.  
26     apply (idpath (S m)).  
27   - cbn.  
28     apply maponpaths.  
29     exact IHn.  
30   Defined.  
31  
32 Proposition comm (m n : nat) : m + n = n + m.  
33 Proof.  
34   induction n as [ | n IHn].  
35   - cbn.  
36     rewrite (plus0 m).  
37     exact (idpath m).  
38   - cbn.
```

The proof view window on the right shows the goal `n : nat` with a hint `(1/1)` and the type `nat`.

The bottom status bar indicates: 0.0.0 21 Loading Coq ✓ coq | demo.v Ln 6, Col 4 Spaces: 2 UTF-8 LF Coq



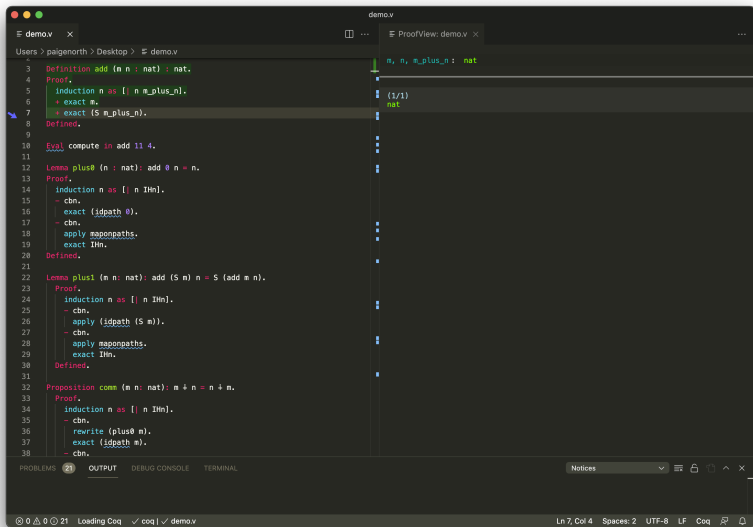
# Type theory



The screenshot shows the Coq IDE with a file named `demo.v` open. The editor displays a proof script for defining addition and proving its properties. The script includes a definition of `add`, a lemma `plus0`, a lemma `plus1`, and a proposition `comm`. The right-hand pane shows the proof view for `demo.v`, indicating that there are no unfocused goals. The bottom status bar shows the Coq version (0.0.0), the file name (`demo.v`), and the current cursor position (Ln 6, Col 13).

```
1  
2  
3 Definition add (n m : nat) : nat.  
4   Proof.  
5     induction n as [ | n m_plus_n].  
6     + exact m.  
7     + exact (S m_plus_n).  
8   Defined.  
9  
10 Eval compute in add 11 4.  
11  
12 Lemma plus0 (n : nat) : add 0 n = n.  
13   Proof.  
14     induction n as [ | n IHn].  
15     - cbn.  
16     - exact (idpath 0).  
17     - cbn.  
18     - apply maponpaths.  
19       exact IHn.  
20   Defined.  
21  
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).  
23   Proof.  
24     induction n as [ | n IHn].  
25     - cbn.  
26     - apply (idpath (S m)).  
27     - cbn.  
28     - apply maponpaths.  
29       exact IHn.  
30   Defined.  
31  
32 Proposition comm (m n : nat) : m + n = n + m.  
33   Proof.  
34     induction n as [ | n IHn].  
35     - cbn.  
36     - rewrite (plus0 m).  
37     - exact (idpath m).  
38     - cbn.
```

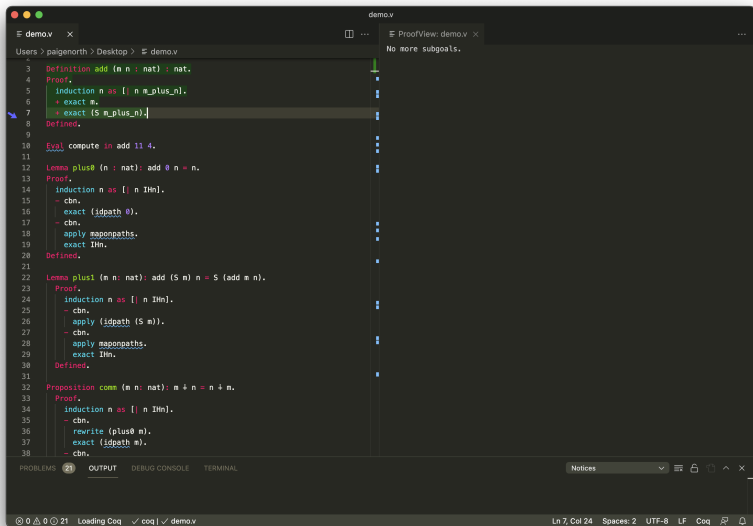
# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers. The script defines an `add` function, proves a lemma `plus0`, a lemma `plus1`, and a proposition `comm`. The right-hand pane shows the proof state for the current goal, which is `n, n, m_plus_n : nat` with a goal of `(1/1) nat`. The bottom status bar indicates the Coq version (0.0.0) and the current file (`demo.v`).

```
1 Definition add (n n' : nat) : nat.  
2   Proof.  
3     induction n as [ | n m_plus_n].  
4     + exact m.  
5     + exact (S m_plus_n).  
6   Defined.  
7  
8 Eval compute in add 11 4.  
9  
10  
11  
12 Lemma plus0 (n : nat) : add 0 n = n.  
13 Proof.  
14   induction n as [ | n IHn].  
15   - cbn.  
16     exact (idpath 0).  
17   - cbn.  
18     apply maponpaths.  
19     exact IHn.  
20 Defined.  
21  
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).  
23 Proof.  
24   induction n as [ | n IHn].  
25   - cbn.  
26     apply (idpath (S m)).  
27   - cbn.  
28     apply maponpaths.  
29     exact IHn.  
30 Defined.  
31  
32 Proposition comm (m n : nat) : m + n = n + m.  
33 Proof.  
34   induction n as [ | n IHn].  
35   - cbn.  
36     rewrite (plus0 m).  
37     exact (idpath m).  
38   - cbn.
```

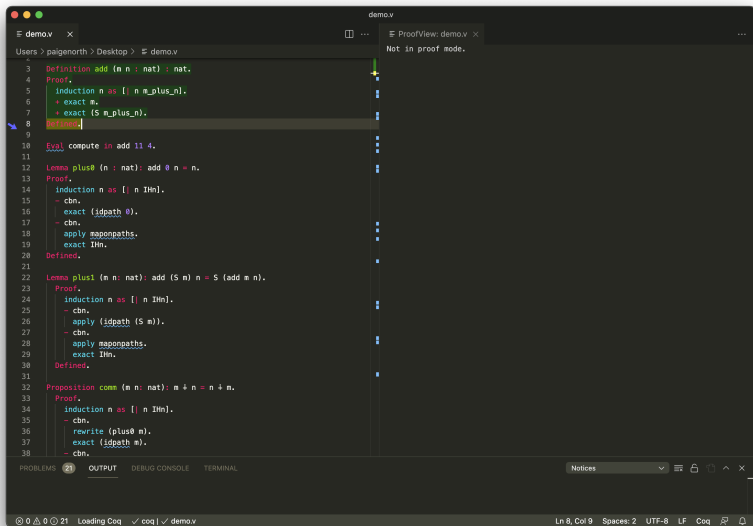
# Type theory



The screenshot shows the Coq IDE with a file named `demo.v`. The editor displays a proof script for natural numbers. The script defines an `add` function, proves lemmas `plus0` and `plus1`, and proves the commutativity of addition (`comm`). The interface includes a top bar with the file name, a toolbar, and a sidebar with tabs for `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, and `TERMINAL`. The `PROBLEMS` tab is active, showing a list of goals. The status bar at the bottom indicates the current position (Ln 7, Col 24), spacing (Spaces: 2), encoding (UTF-8), line feed (LF), and the loaded file (`demo.v`).

```
1  
2  
3 Definition add (n m : nat) : nat.  
4   Proof.  
5     induction n as [ | n m_plus_n].  
6     + exact m.  
7     + exact (S m_plus_n).  
8   Defined.  
9  
10 Eval compute in add 11 4.  
11  
12 Lemma plus0 (n : nat) : add 0 n = n.  
13   Proof.  
14     induction n as [ | n IHn].  
15     - cbn.  
16     - exact (idpath 0).  
17     - cbn.  
18     - apply maponpaths.  
19       exact IHn.  
20   Defined.  
21  
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).  
23   Proof.  
24     induction n as [ | n IHn].  
25     - cbn.  
26     - apply (idpath (S m)).  
27     - cbn.  
28     - apply maponpaths.  
29       exact IHn.  
30   Defined.  
31  
32 Proposition comm (m n : nat) : m + n = n + m.  
33   Proof.  
34     induction n as [ | n IHn].  
35     - cbn.  
36     - rewrite (plus0 m).  
37     - exact (idpath m).  
38     - cbn.
```

# Type theory



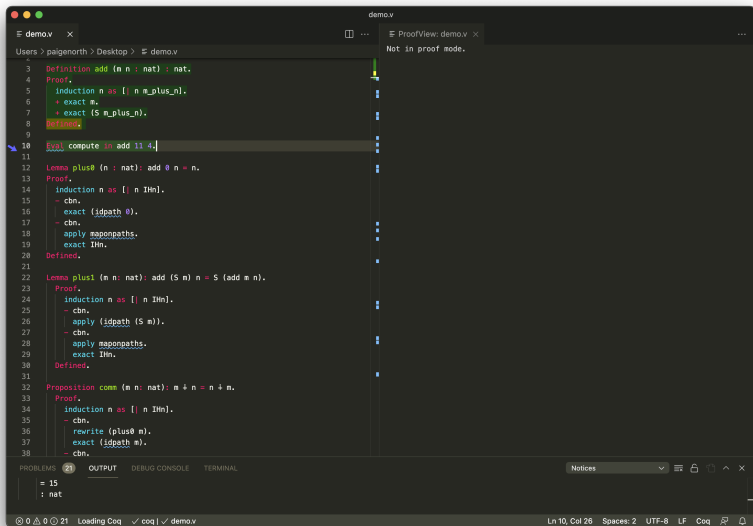
The screenshot shows a Coq IDE window titled "demo.v". The editor displays a proof script for natural numbers. The script defines an addition function, proves its correctness for zero and successor cases, and proves the commutative property. The interface includes a file explorer, a proof view pane, and a status bar at the bottom.

```
1 Users > paigenorth > Desktop > demo.v
2
3 Definition add (n m : nat) : nat.
4   Proof.
5     induction n as [ | n m_plus_n].
6     + exact m.
7     + exact (S m_plus_n).
8   Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat) : add 0 n = n.
13   Proof.
14     induction n as [ | n IHn].
15     - cbn.
16     - exact (idpath 0).
17     - cbn.
18     - apply maponpaths.
19     - exact IHn.
20   Defined.
21
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).
23   Proof.
24     induction n as [ | n IHn].
25     - cbn.
26     - apply (idpath (S m)).
27     - cbn.
28     - apply maponpaths.
29     - exact IHn.
30   Defined.
31
32 Proposition comm (m n : nat) : m + n = n + m.
33   Proof.
34     induction n as [ | n IHn].
35     - cbn.
36     - rewrite (plus0 m).
37     - exact (idpath m).
38     - cbn.
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL Notices

0.0.0 21 Loading Coq ✓ coq | demo.v Ln 8, Col 9 Spaces: 2 UTF-8 LF Coq

# Type theory



The screenshot shows a Coq IDE window titled "demo.v". The editor displays a proof script for natural numbers. The script defines an addition function, proves its correctness with lemmas, and states a commutativity proposition. The bottom status bar indicates the file is being loaded by Coq.

```
demo.v
Users > paigenorth > Desktop > demo.v
3 Definition add (n m : nat) : nat.
4   Proof.
5     induction n as [ | n m_plus_n].
6     + exact m.
7     + exact (S m_plus_n).
8   Defined.
9
10 Eval compute in add 11 4.
11
12 Lemma plus0 (n : nat) : add 0 n = n.
13 Proof.
14   induction n as [ | n IHn].
15   - cbn.
16     exact (idpath 0).
17   - cbn.
18     apply maponpaths.
19     exact IHn.
20 Defined.
21
22 Lemma plus1 (m n : nat) : add (S m) n = S (add m n).
23 Proof.
24   induction n as [ | n IHn].
25   - cbn.
26     apply (idpath (S m)).
27   - cbn.
28     apply maponpaths.
29     exact IHn.
30 Defined.
31
32 Proposition comm (m n : nat) : m + n = n + m.
33 Proof.
34   induction n as [ | n IHn].
35   - cbn.
36     rewrite (plus0 m).
37     exact (idpath m).
38   - cbn.
```

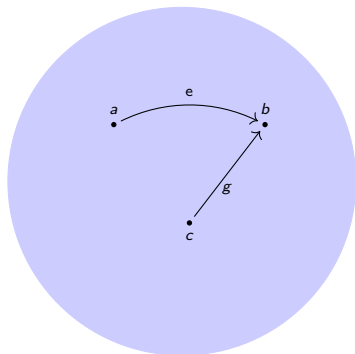
PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL

= 15  
: nat

Ln 10, Col 26 Spaces: 2 UTF-8 LF Coq

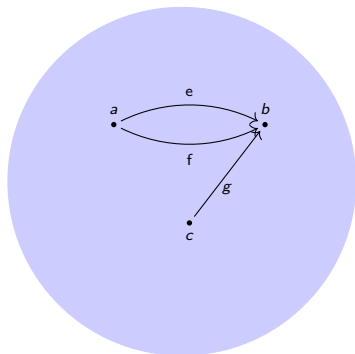
# Homotopy type theory

- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .



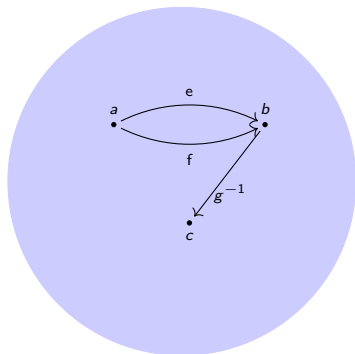
# Homotopy type theory

- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .
- ▶ We can have equalities  $e, f : a = b$ .



# Homotopy type theory

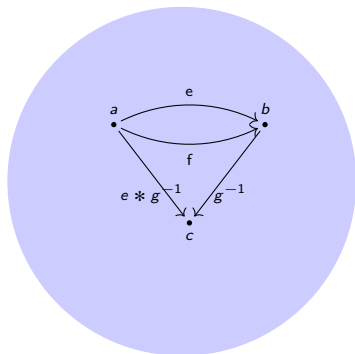
- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .
- ▶ We can have equalities  $e, f : a = b$ .
- ▶ Equalities are invertible.





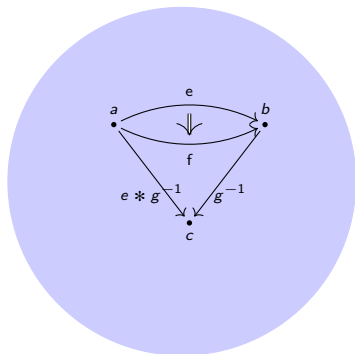
# Homotopy type theory

- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .
- ▶ We can have equalities  $e, f : a = b$ .
- ▶ Equalities are invertible.
- ▶ Equalities are composable.



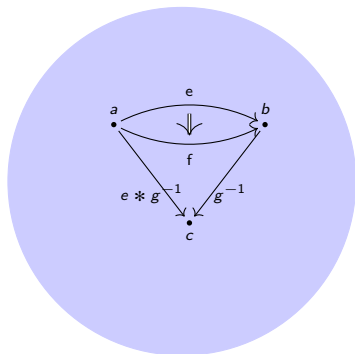
# Homotopy type theory

- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .
- ▶ We can have equalities  $e, f : a = b$ .
- ▶ Equalities are invertible.
- ▶ Equalities are composable.
- ▶ There can be “higher” equalities.



# Homotopy type theory

- ▶ Equality is also given inductively.
- ▶ The **equality type**  $a = b$  (for two terms  $a, b : A$ ) is generated inductively by the *canonical term*  $r_a : a = a$  for each term  $a : A$ .
  - ▶ Just as  $\mathbb{N}$  is generated by the canonical elements  $0 : \mathbb{N}$  and  $Sn : \mathbb{N}$  for each  $n : \mathbb{N}$ .
- ▶ We can have equalities  $e, f : a = b$ .
- ▶ Equalities are invertible.
- ▶ Equalities are composable.
- ▶ There can be “higher” equalities.
- ▶ This makes types behave like homotopy types or spaces.



# Interpretation of type theory in homotopy type theory

- ▶ Homotopy type theory has a rigorous interpretation in the category of simplicial sets (among others), the category in which classical homotopy theory takes place.<sup>5</sup>
- ▶ We can check and develop the mathematics of homotopy types / spaces (in particular, simplicial sets) in homotopy type theory.
  - ▶ Homotopy groups of spheres<sup>6</sup>
  - ▶ Higher groups<sup>7</sup>
  - ▶ etc
- ▶ The **Univalence Axiom** allows us to treat equivalent things as equal.
  - ▶ Different implementations of programs (with different advantages) can be equated.<sup>8</sup>

---

<sup>5</sup>Lumsdaine, Kapulkin, Voevodsky 2012

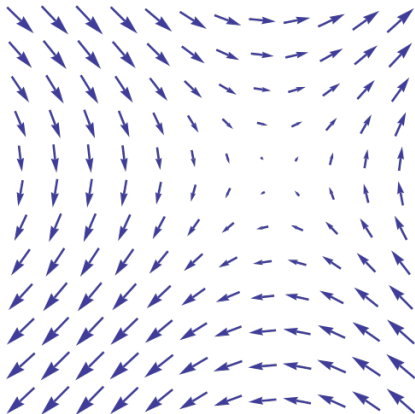
<sup>6</sup>Licata, Shulman, Brunerie

<sup>7</sup>Buchholz, van Doorn, Rijke, 2018

<sup>8</sup>Angiuli, Cavallo, Mörtberg, Zeuner 2020

# Directed spaces

- ▶ Given a manifold  $\mathcal{M}$  with vector field  $\mathcal{F}$ , the manifold  $\mathcal{M}$  together with the finite-time trajectories<sup>9</sup> produces a directed space.

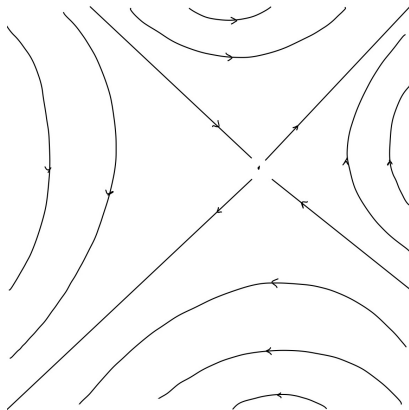


---

<sup>9</sup>closed under reparametrization

# Directed spaces

- ▶ Given a manifold  $\mathcal{M}$  with vector field  $\mathcal{F}$ , the manifold  $\mathcal{M}$  together with the finite-time trajectories<sup>9</sup> produces a directed space.



---

<sup>9</sup>closed under reparametrization

## Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former `hom` on top of a modal type theory with modal transformations `op`, `core`.

---

<sup>10</sup>Under construction, see North 2019

## Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former  $\text{hom}$  on top of a modal type theory with modal transformations  $\text{op}$ ,  $\text{core}$ .
- ▶ For each type  $A$  and each  $a : A$ ,  $b : A$ , we have  $\text{hom}(a, b)$ .

---

<sup>10</sup>Under construction, see North 2019



## Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former  $\text{hom}$  on top of a modal type theory with modal transformations  $\text{op}$ ,  $\text{core}$ .
- ▶ For each type  $A$  and each  $a : \text{op } A, b : A$ , we have  $\text{hom}(a, b)$ .
- ▶ There is one canonical element  $1_a : \text{hom}(a, a)$ ,

---

<sup>10</sup>Under construction, see North 2019

# Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former  $\text{hom}$  on top of a modal type theory with modal transformations  $\text{op}$ ,  $\text{core}$ .
- ▶ For each type  $A$  and each  $a : \text{op } A, b : A$ , we have  $\text{hom}(a, b)$ .
- ▶ There is one canonical element  $1_a : \text{hom}(a, a)$ ,
- ▶ But it has **two** induction principles: a forward and a backward one.

---

<sup>10</sup>Under construction, see North 2019

# Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former  $\text{hom}$  on top of a modal type theory with modal transformations  $\text{op}$ ,  $\text{core}$ .
- ▶ For each type  $A$  and each  $a : {}^{\text{op}} A$ ,  $b : A$ , we have  $\text{hom}(a, b)$ .
- ▶ There is one canonical element  $1_a : \text{hom}(a, a)$ ,
- ▶ But it has **two** induction principles: a forward and a backward one.
- ▶ In a category, directed space, etc, given a homomorphism  $f : x \rightarrow y$ , there are two 'homomorphisms' from one of the form  $1_a$  to it.

$$\begin{array}{ccc} x & \xlongequal{1_x} & x \\ \parallel 1_x & & \downarrow f \\ x & \xrightarrow{f} & y \end{array} \qquad \begin{array}{ccc} y & \xleftarrow{f} & x \\ \parallel 1_y & & \downarrow f \\ y & \xlongequal{1_y} & y \end{array}$$

---

<sup>10</sup>Under construction, see North 2019

# Directed homotopy type theory<sup>10</sup>

- ▶ We introduce a **homomorphism** type former  $\text{hom}$  on top of a modal type theory with modal transformations  $\text{op}$ ,  $\text{core}$ .
- ▶ For each type  $A$  and each  $a : {}^{\text{op}} A$ ,  $b : A$ , we have  $\text{hom}(a, b)$ .
- ▶ There is one canonical element  $1_a : \text{hom}(a, a)$ ,
- ▶ But it has **two** induction principles: a forward and a backward one.
- ▶ In a category, directed space, etc, given a homomorphism  $f : x \rightarrow y$ , there are two 'homomorphisms' from one of the form  $1_a$  to it.

$$\begin{array}{ccc} x & \xrightarrow{1_x} & x \\ \parallel 1_x & & \downarrow f \\ x & \xrightarrow{f} & y \end{array} \qquad \begin{array}{ccc} y & \xleftarrow{f} & x \\ \parallel 1_y & & \downarrow f \\ y & \xleftarrow{1_y} & y \end{array}$$

- ▶ This has an interpretation<sup>10</sup> in the category of categories, categories of directed spaces, etc...

---

<sup>10</sup>Under construction, see North 2019

## Future work

- ▶ We hope to develop this type theory.
- ▶ Check theorems from directed homotopy theory, dynamics, etc.
- ▶ Develop higher inductive types. These correspond to directed homotopy colimits in some cases and perhaps a notion of hybrid fold.

Thank you!