

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 theory² is a foundation for the study of **directed** homotopy types (**directed** topological spaces).
- ▶ Directed spaces³ capture much of the theory of vector fields on manifolds⁴

¹Curry-Howard correspondence

²under construction

³See Sanjeevi Krishnan's talk for more details about directed spaces.

⁴See 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 a code editor interface with two tabs: 'demo.v' and 'ProofView: demo.v'. The 'demo.v' tab contains Coq code defining addition and related lemmas. The 'ProofView: demo.v' tab shows the proof state for the 'plus0' lemma, indicating it is not in proof mode.

```
demo.v
Users > paigenorth > Desktop > demo.v
Not in proof mode.

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

PROBLEMS: 21 OUTPUT DEBUG CONSOLE TERMINAL Notices

① ② ③ ④ ⑤ 21 Loading Coq ✓ coq | ✓ demo.v

Ln 1, Col 40 Spaces: 2 UTF-8 LF Coq ⌂ ⌃ ⌄

Type theory

The screenshot shows a Coq development environment with two panes. The left pane displays the source code of a file named `demo.v`. The right pane shows the proof state, which is currently empty.

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

1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13     - cbn.
14     + exact (idpath 0).
15   - cbn.
16   apply maponpaths.
17   exact IHn.
18 Defined.
19
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23     - cbn.
24     + apply (idpath (S m)).
25   - cbn.
26   apply maponpaths.
27   exact IHn.
28 Defined.
29
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33     - cbn.
34     + rewrite (plus0 m).
35     | exact (idpath m).
36   - cbn.
```

Below the code editor, there is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The OUTPUT tab is selected. At the bottom of the interface, there is a status bar showing build progress (21), loading status (Loading Coq), and file information (Ln 3, Col 34, Spaces: 2, UTF-8, LF, Coq).

Type theory

The screenshot shows a Coq development environment with two main panes: a code editor and a proof view.

Code Editor (Left Pane):

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

Proof View (Right Pane):

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

Bottom Bar:

- PROBLEMS: 21
- OUTPUT
- DEBUG CONSOLE
- TERMINAL
- Notices
- ☰
- ✖

Status Bar:

⌚ ⚡ ⚡ ⚡ 21 Loading Coq ✓ coq | ✓ demo.v Line 4, Col 7 Spaces: 2 UTF-8 LF Coq ⌂ ⌂ ⌂ ⌂

Type theory

The screenshot shows a Coq development environment with two tabs: "demo.v" and "ProofView: demo.v".

The "demo.v" tab displays a Coq script with the following content:

```
1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13     - cbn.
14     + exact (idpath 0).
15   - cbn.
16   apply maponpaths.
17   exact IHn.
18 Defined.
19
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23     - cbn.
24     + apply (idpath (S m)).
25   - cbn.
26   apply maponpaths.
27   exact IHn.
28 Defined.
29
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33     - cbn.
34     + rewrite (plus0 m).
35     | exact (idpath m).
36   - cbn.
```

The "ProofView: demo.v" tab shows the state of the proof. It indicates that the variable `m` has type `nat`. The proof tree shows two branches:

- (1/2) `nat`
- (2/2) `nat`

The bottom of the interface shows the "PROBLEMS" section with 21 errors, the "OUTPUT" tab selected, and the status bar indicating "Ln 5, Col 33 Spaces: 2 UTF-8 LF Coq".

Type theory

The screenshot shows a Coq development environment with two main panes: a code editor on the left and a proof view on the right.

Code Editor (demo.v):

```
1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13   | cbn.
14   + exact (idpath 0).
15   - cbn.
16   apply maponpaths.
17   exact IHn.
18 Defined.
19
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23   | cbn.
24   + apply (idpath (S m)).
25   - cbn.
26   apply maponpaths.
27   exact IHn.
28 Defined.
29
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33   | cbn.
34   + rewrite (plus0 m).
35   - exact (idpath m).
36   - cbn.
```

Proof View (demo.v):

The proof view shows the goal $n : \text{nat}$ and its derivation:

- (1/1)
- nat

Below the proof view, there are several status indicators and a navigation bar:

- PROBLEMS: 21
- OUTPUT
- DEBUG CONSOLE
- TERMINAL
- Notices
- File, Edit, View, Tools, Help

At the bottom, there are icons for saving, closing, and other file operations.

Type theory

The screenshot shows a Coq development environment with two main panes. The left pane displays the source code for a file named `demo.v`. The right pane shows the proof view for the same file.

Left Pane (demo.v):

```
demo.v
Users > paigenorth > Desktop > demo.v
1 Definition add {m 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 Lemma plus0 {n : nat}: add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13   - cbn.
14   - exact (idpath 0).
15   - cbn.
16   - apply maponpaths.
17   - exact IHn.
18 Defined.
19 
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23   - cbn.
24   - apply (idpath (S m)).
25   - cbn.
26   - apply maponpaths.
27   - exact IHn.
28 Defined.
29 
30 Proposition comm {m n: nat}: m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33   - cbn.
34   - rewrite (plus0 m).
35   - exact (idpath m).
36   - cbn.
```

Right Pane (ProofView):

There are unfocused goals.

At the bottom of the interface, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The OUTPUT tab is currently selected. At the bottom right, there are notices and icons for saving, closing, and other operations.

Type theory

The screenshot shows a Coq development environment with two panes. The left pane displays the source file `demo.v` containing Coq code. The right pane shows the proof view for the current goal.

File Content (`demo.v`):

```
1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13   - cbn.
14   + exact (idpath 0).
15   - cbn.
16   + apply maponpaths.
17   + exact IHn.
18 Defined.
19
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23   - cbn.
24   + apply (idpath (S m)).
25   - cbn.
26   + apply maponpaths.
27   + exact IHn.
28 Defined.
29
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33   - cbn.
34   + rewrite (plus0 m).
35   + exact (idpath m).
36   - cbn.
```

Proof View:

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

At the bottom, the status bar indicates: `① 0 ③ 0 ④ 21 Loading Coq ✓ coq | ✓ demo.v`, `Ln 7, Col 4 Spaces: 2 UTF-8 LF Coq`, and `4/10`.

Type theory

The screenshot shows a Coq development environment with two tabs: "demo.v" and "ProofView: demo.v".

demo.v:

```
1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13   * cbn.
14   * exact (idpath 0).
15   * cbn.
16   * apply maponpaths.
17   * exact IHn.
18 Defined.
19
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23   * cbn.
24   * apply (idpath (S m)).
25   * cbn.
26   * apply maponpaths.
27   * exact IHn.
28 Defined.
29
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33   * cbn.
34   * rewrite (plus0 m).
35   * exact (idpath m).
36   * cbn.
```

ProofView: demo.v:

No more subgoals.

At the bottom of the interface, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, Notices, and other icons.

Type theory

The screenshot shows a code editor with two tabs: "demo.v" and "ProofView: demo.v". The "demo.v" tab contains Coq code defining addition and related lemmas. The "ProofView: demo.v" tab shows the proof state for the "plus0" lemma.

```
demo.v
Users > paigenorth > Desktop > demo.v
Not in proof mode.

1 Definition add (m 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 Lemma plus0 (n : nat): add 0 n = n.
11 Proof.
12   induction n as [| n IHn].
13   - cbn.
14   - exact (idpath 0).
15   - cbn.
16   - apply maponpaths.
17   - exact IHn.
18 Defined.
19 
20 Lemma plus1 {m n: nat}: add (S m) n = S (add m n).
21 Proof.
22   induction n as [| n IHn].
23   - cbn.
24   - apply (idpath (S m)).
25   - cbn.
26   - apply maponpaths.
27   - exact IHn.
28 Defined.
29 
30 Proposition comm (m n: nat): m + n = n + m.
31 Proof.
32   induction n as [| n IHn].
33   - cbn.
34   - rewrite (plus0 m).
35   - exact (idpath m).
36   - cbn.
```

PROBLEMS: 21 OUTPUT DEBUG CONSOLE TERMINAL Notices

① ② ③ ④ ⑤ 21 Loading Coq ✓ coq | ✓ demo.v Ln 8, Col 9 Spaces: 2 UTF-8 LF Coq ⌂ ⌃ ⌄

Type theory

```
demo.v
demo.v x
ProofView: demo.v x
Not in proof mode.

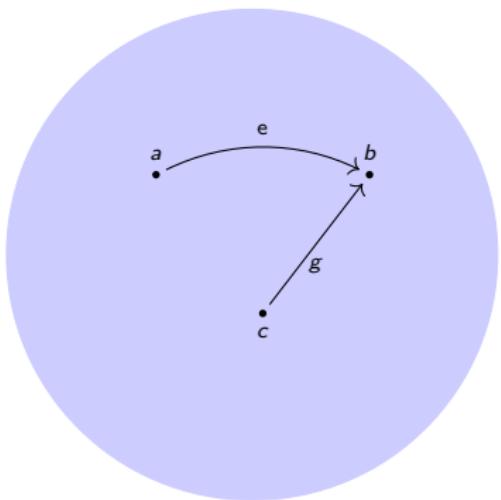
Users > paigenorth > Desktop > demo.v

3 Definition add (m n : nat) : nat.
4 Proof.
5 induction m as [| m 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 m as [| m IHm].
25   - cbn.
26   | apply (idpath (S m)).
27   - cbn.
28   apply maponpaths.
29   exact IHm.
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
= 15 : nat
Ln 10, Col 26 Spaces: 2 UTF-8 LF Cog
```

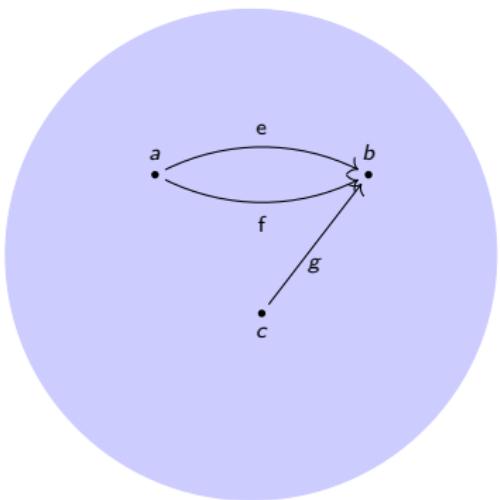
Homotopy type theory

- ▶ 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 $n : \mathbb{N}$.



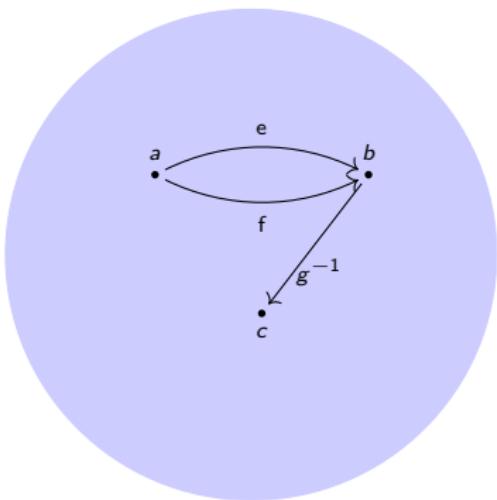
Homotopy type theory

- ▶ 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 $e, f : a = b$.



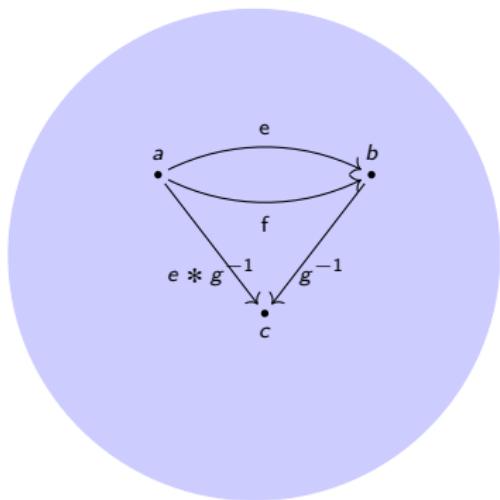
Homotopy type theory

- ▶ 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.



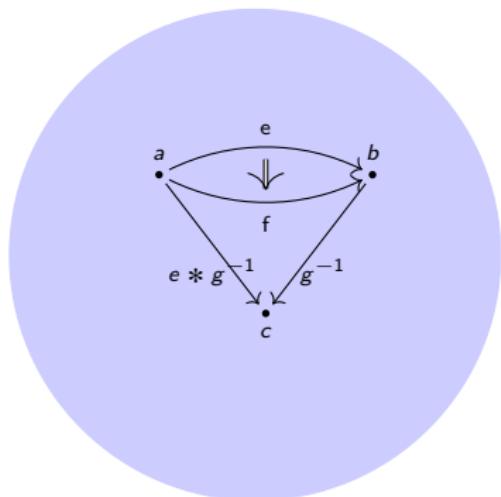
Homotopy type theory

- ▶ 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.



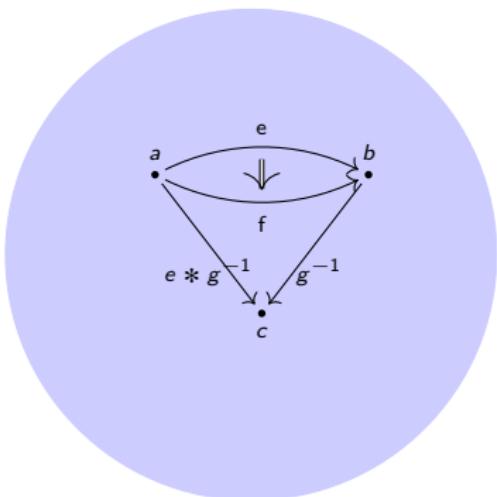
Homotopy type theory

- ▶ 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.⁵
- ▶ We can check and develop the mathematics of homotopy types / spaces (in particular, simplicial sets) in homotopy type theory.
 - ▶ Homotopy groups of spheres⁶
 - ▶ Higher groups⁷
 - ▶ etc
- ▶ The **Univalence Axiom** allows us to treat equivalent things as equal.
 - ▶ Different implementations of programs (with different advantages) can be equated.⁸

⁵Lumsdaine, Kapulkin, Voevodsky 2012

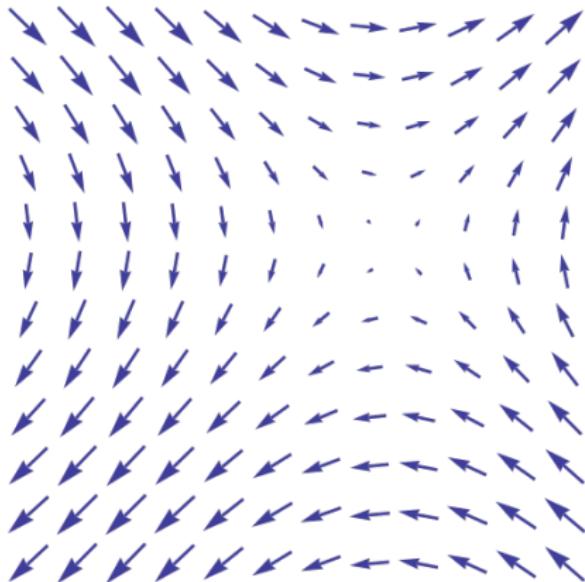
⁶Licata, Shulman, Brunerie

⁷Buchholz, van Doorn, Rijke, 2018

⁸Angiuli, Cavallo, Mörtberg, Zeuner 2020

Directed spaces

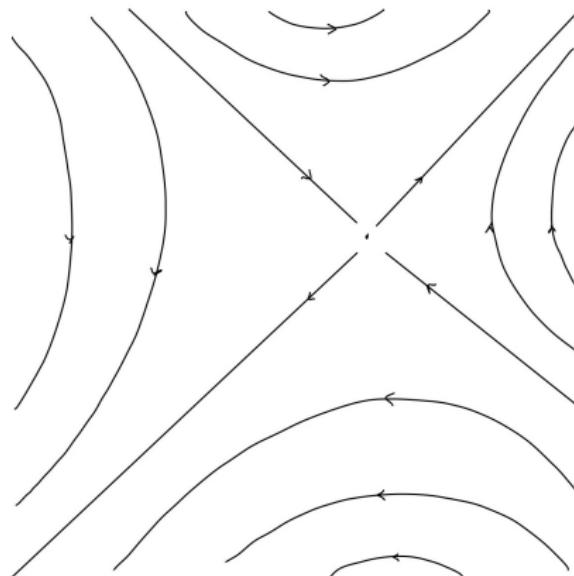
- Given a manifold \mathcal{M} with vector field \mathcal{F} , the manifold \mathcal{M} together with the finite-time trajectories⁹ produces a directed space.



⁹closed under reparametrization

Directed spaces

- Given a manifold \mathcal{M} with vector field \mathcal{F} , the manifold \mathcal{M} together with the finite-time trajectories⁹ produces a directed space.



⁹closed under reparametrization

Directed homotopy type theory¹⁰

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

¹⁰Under construction, see North 2019

Directed homotopy type theory¹⁰

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

¹⁰Under construction, see North 2019

Directed homotopy type theory¹⁰

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

¹⁰Under construction, see North 2019

Directed homotopy type theory¹⁰

- ▶ We introduce a **homomorphism** type former hom on top of a modal type theory with modal transformations op , core .
- ▶ For each type A and each $a : A$, $b : A$, 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.

¹⁰Under construction, see North 2019

Directed homotopy type theory¹⁰

- ▶ We introduce a **homomorphism** type former hom on top of a modal type theory with modal transformations op , core .
- ▶ For each type A and each $a : A, b : A$, 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.

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

¹⁰Under construction, see North 2019

Directed homotopy type theory¹⁰

- ▶ We introduce a **homomorphism** type former hom on top of a modal type theory with modal transformations op , core .
- ▶ For each type A and each $a : A, b : A$, 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.

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

- ▶ This has an interpretations¹⁰ in the category of categories, categories of directed spaces, etc...

¹⁰Under 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 hybrifold.

Thank you!