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[2] is a foundation for the study of **directed** homotopy types (**directed** topological spaces).
- Directed spaces[3] capture much of the theory of vector fields on manifolds[4]

[1] Curry-Howard correspondence

[2] under construction

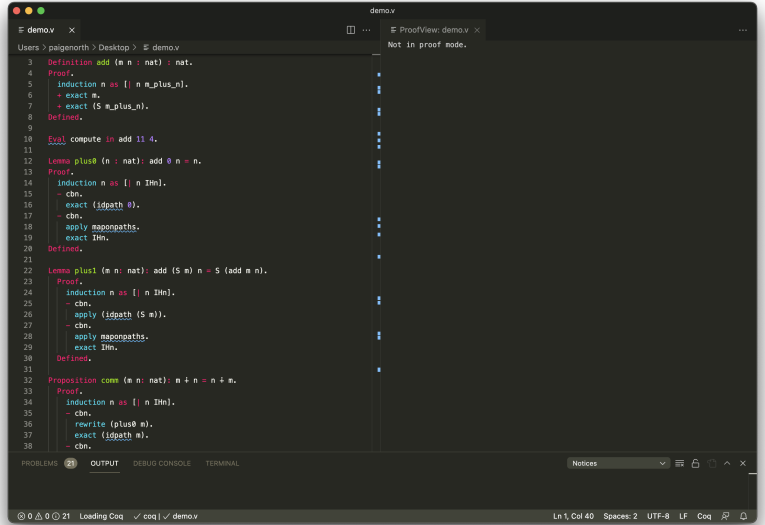
[3] See Sanjeevi Krishnan's talk for more details about directed spaces.

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

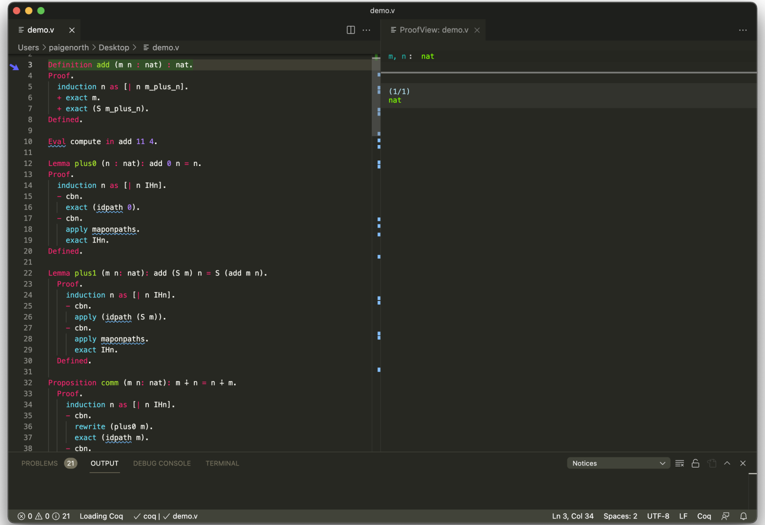
Type theory

- 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 $A \times B$, $A + B$, $A \to B$, 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 $n : \mathbb{N}$.

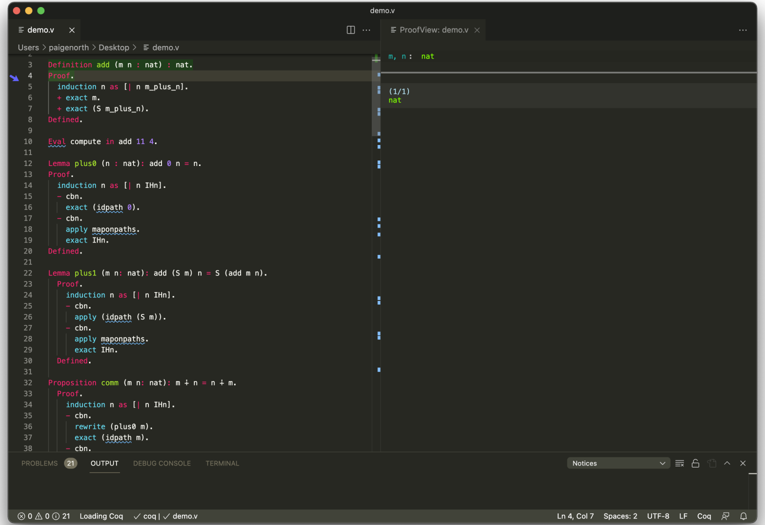
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Not in proof mode.
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m ∔ n = n ∔ m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq
coq | demo.v
Ln 1, Col 40
Spaces: 2
UTF-8
LF
Coq

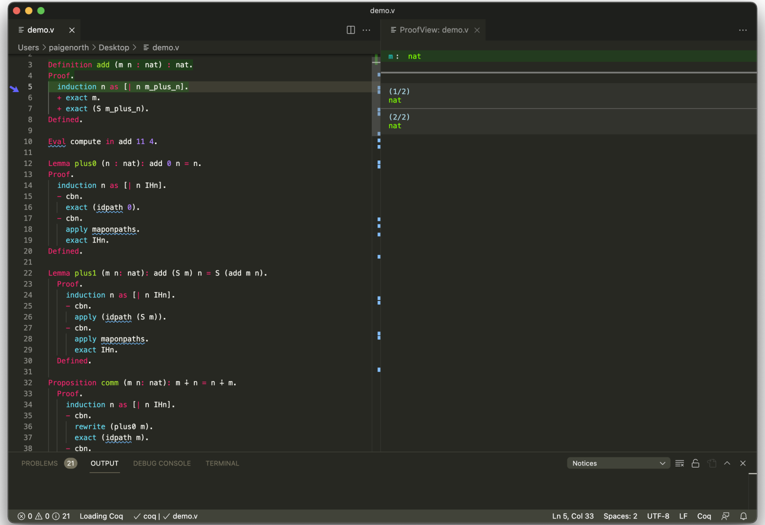
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
m, n : nat
(1/1)
nat
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq
coq | demo.v
Ln 3, Col 34 Spaces: 2 UTF-8 LF Coq

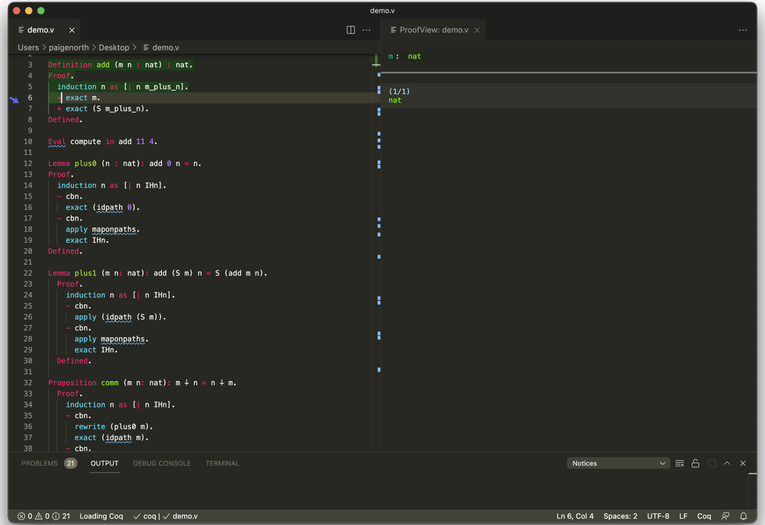
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
m, n : nat
(1/1)
nat
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq coq | demo.v
Ln 4, Col 7 Spaces: 2 UTF-8 LF Coq

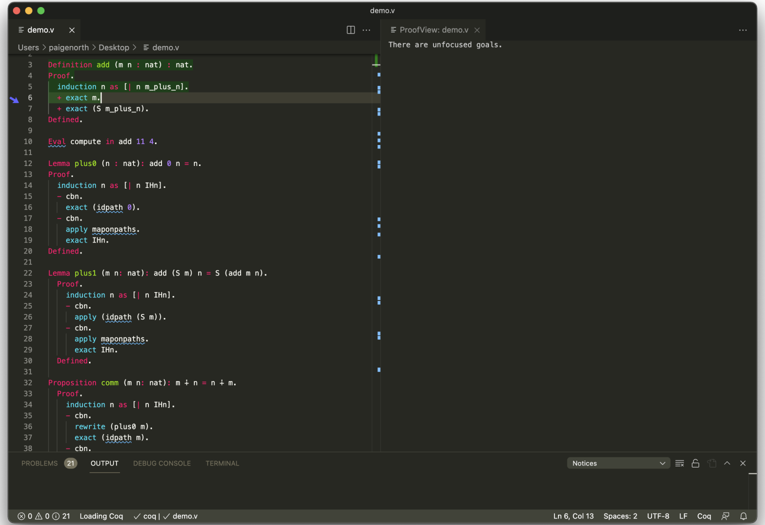
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
m : nat
(1/2)
nat
(2/2)
nat
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq
coq | demo.v
Ln 5, Col 33
Spaces: 2
UTF-8
LF
Coq

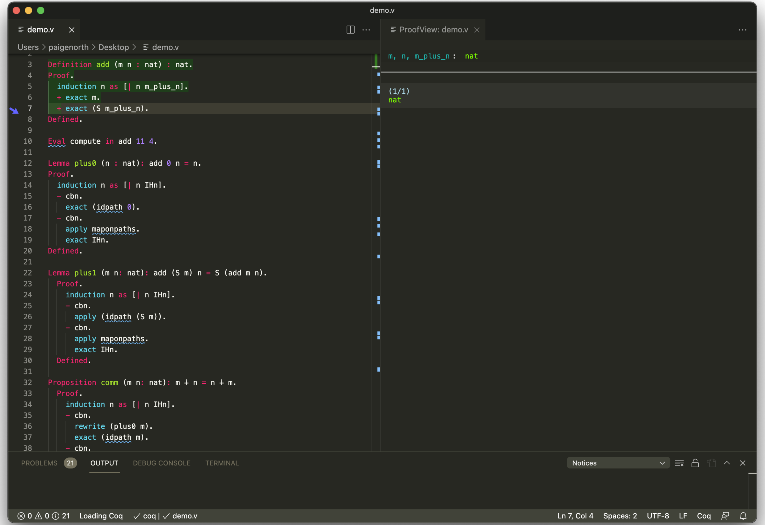
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
m : nat
(1/1)
nat
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq
coq | demo.v
Ln 6, Col 4 Spaces: 2 UTF-8 LF Coq

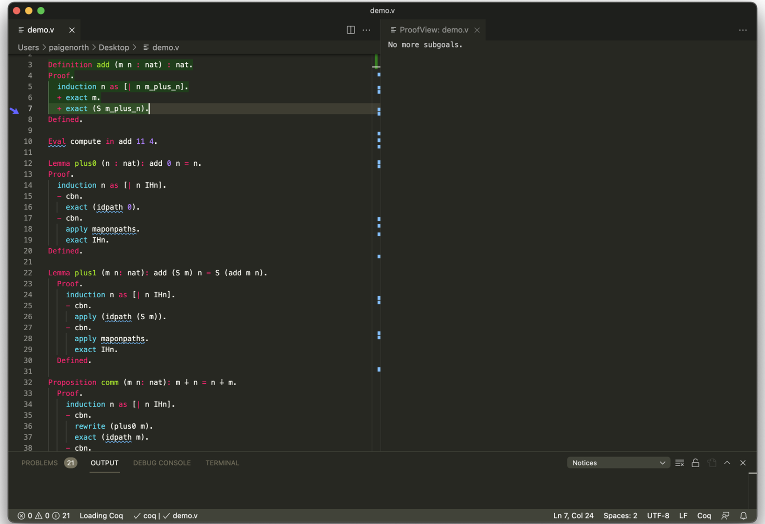
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
There are unfocused goals.
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq
coq | demo.v
Ln 6, Col 13 Spaces: 2 UTF-8 LF Coq

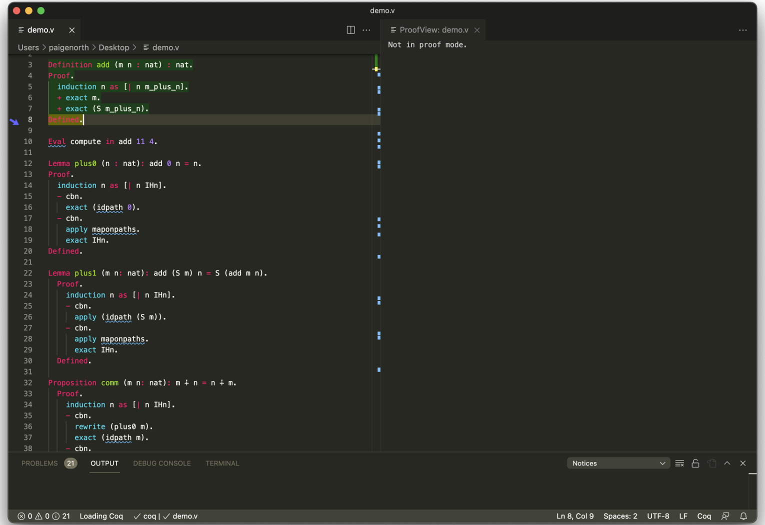
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
m, n, m_plus_n : nat
(1/1)
nat
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21
Loading Coq
coq | demo.v
Ln 7, Col 4
Spaces: 2
UTF-8
LF
Coq

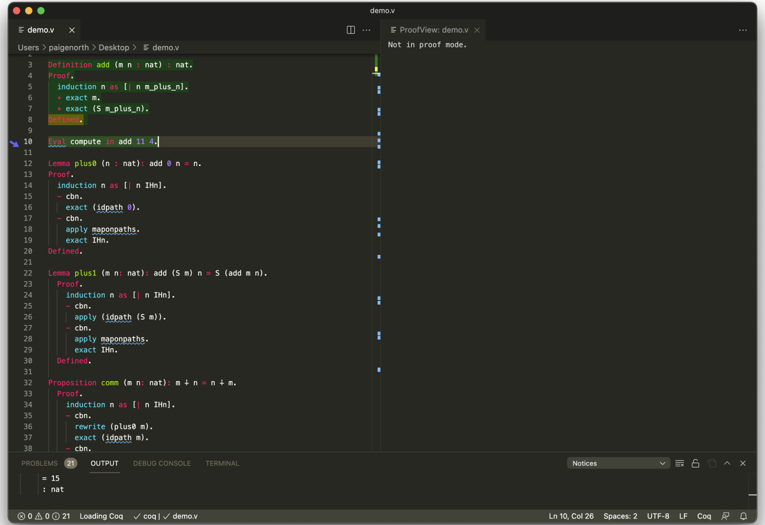
Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
No more subgoals.
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq coq | demo.v
Ln 7, Col 24 Spaces: 2 UTF-8 LF Coq

Type theory

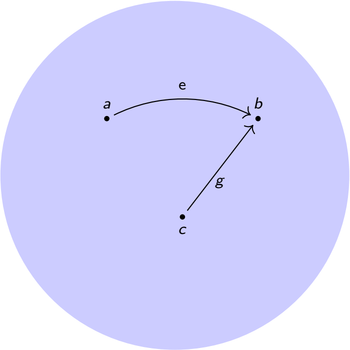

demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
Not in proof mode.
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
0 0 21 Loading Coq ✓ coq | ✓ demo.v
Ln 8, Col 9 Spaces: 2 UTF-8 LF Coq

Type theory


demo.v
demo.v
ProofView: demo.v
Users > paigenorth > Desktop > demo.v
Not in proof mode.
Definition add (m n : nat) : nat.
Proof.
induction n as [| n m_plus_n].
+ exact m.
+ exact (S m_plus_n).
Defined.
Eval compute in add 11 4.
Lemma plus0 (n : nat): add 0 n = n.
Proof.
induction n as [| n IHn].
- cbn.
exact (idpath 0).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Lemma plus1 (m n: nat): add (S m) n = S (add m n).
Proof.
induction n as [| n IHn].
- cbn.
apply (idpath (S m)).
- cbn.
apply maponpaths.
exact IHn.
Defined.
Proposition comm (m n: nat): m + n = n + m.
Proof.
induction n as [| n IHn].
- cbn.
rewrite (plus0 m).
exact (idpath m).
- cbn.
PROBLEMS 21
OUTPUT
DEBUG CONSOLE
TERMINAL
Notices
= 15
: nat
0 0 21
Loading Coq
coq | demo.v
Ln 10, Col 26
Spaces: 2
UTF-8
LF
Coq

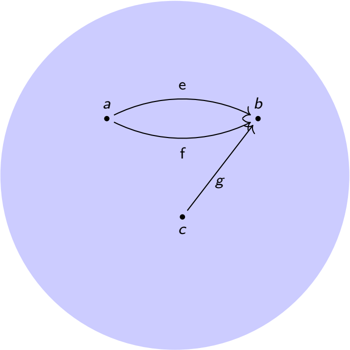
Homotopy type theory

- 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 $n : \mathbb{N}$.



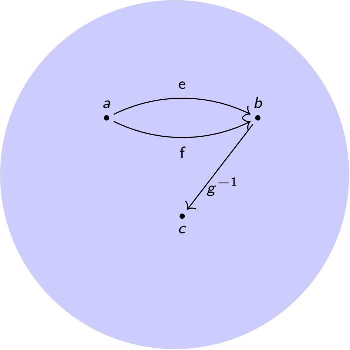
Homotopy type theory

- 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 $e, f : a = b$.



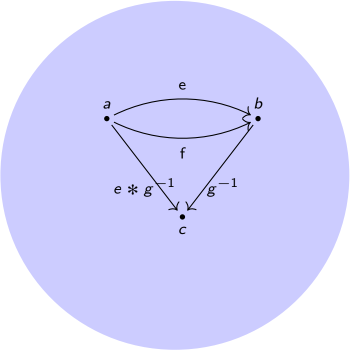
Homotopy type theory

- 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.



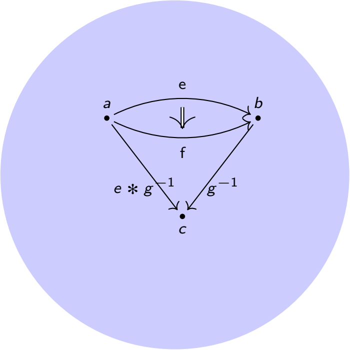
Homotopy type theory

- 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.



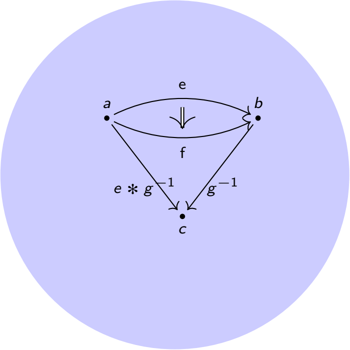
Homotopy type theory

- 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.[5]
- We can check and develop the mathematics of homotopy types / spaces (in particular, simplicial sets) in homotopy type theory.
  - Homotopy groups of spheres[6]
  - Higher groups[7]
  - etc
- The **Univalence Axiom** allows us to treat equivalent things as equal.
  - Different implementations of programs (with different advantages) can be equated.[8]

---

[5] Lumsdaine, Kapulkin, Voevodsky 2012

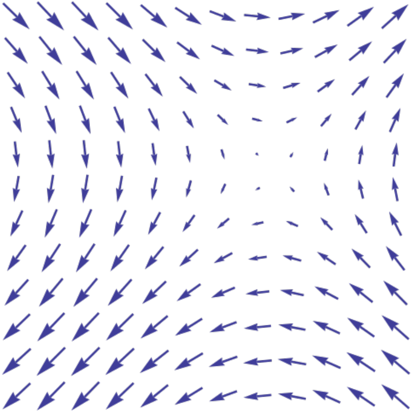
[6] Licata, Shulman, Brunerie

[7] Buchholz, van Doorn, Rijke, 2018

[8] Angiuli, Cavallo, Mörtberg, Zeuner 2020

Directed spaces

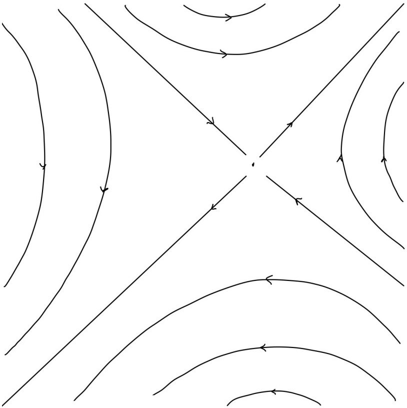
- Given a manifold $\mathcal{M}$ with vector field $\mathcal{F}$, the manifold $\mathcal{M}$ together with the finite-time trajectories[9] produces a directed space.



[9] 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[9] produces a directed space.



[9] closed under reparametrization

Directed homotopy type theory[10]

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

[10] Under construction, see North 2019

Directed homotopy type theory[10]

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

[10] Under construction, see North 2019

Directed homotopy type theory[10]

- 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 $a :^{\mathsf{op}} A, b : A$, we have $\mathrm{hom}(a, b)$.
- There is one canonical element $1_a : \mathrm{hom}(a, a)$,

[10] Under construction, see North 2019

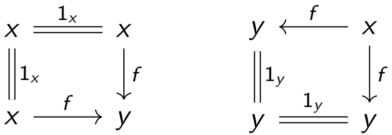
Directed homotopy type theory[10]

- 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 $a :^{\mathsf{op}} A, b : A$, we have $\mathrm{hom}(a, b)$.
- There is one canonical element $1_a : \mathrm{hom}(a, a)$,
- But it has **two** induction principles: a forward and a backward one.

[10] Under construction, see North 2019

Directed homotopy type theory[10]

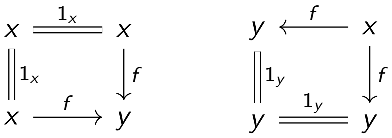
- We introduce a **homomorphism** type former hom on top of a modal type theory with modal transformations op, core.
- For each type A and each $a :^{\mathrm{op}} A, b : A$, we have $\hom(a, b)$.
- There is one canonical element $1_a : \hom(a, a)$,
- But it has **two** induction principles: a forward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



[10] Under construction, see North 2019

Directed homotopy type theory[10]

- 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 $a :^{\mathrm{op}} A, b : A$, we have $\mathrm{hom}(a, b)$.
- There is one canonical element $1_a : \mathrm{hom}(a, a)$,
- But it has **two** induction principles: a forward and a backward one.
- In a category, directed space, etc, given a homomorphism $f : x \to y$, there are two 'homomorphisms' from one of the form 1_a to it.



- This has an interpretations[10] in the category of categories, categories of directed spaces, etc...

[10]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 hybrifold.

Thank you!