CS2309 Presentation: Constructive Mathematics and Computer Programming

Per Martin-Löf 1979

Tan Yee Jian

October 1, 2021

Flow

Motive
- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results
- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Outline

Motive
- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results
- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Motive - Why constructive mathematics?

Imagine the conversation:

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Is there any integer that is even?

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Is there any integer that is even?
- Me: Yes there is.

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Is there any integer that is even?
- Me: Yes there is.
- You: So, what is it?

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Is there any integer that is even?
- Me: Yes there is.
- You: So, what is it?
- Me: I can prove that it exists, but I cannot tell you a specific number.

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Is there any integer that is even?
- Me: Yes there is.
- You: So, what is it?
- Me: I can prove that it exists, but I cannot tell you a specific number.
- You: ???

Motive - Why constructive mathematics?

Definition

Even numbers are integers $x \in \mathbb{Z}$ where $x \equiv 0 \mod 2$.

Theorem

Even numbers exist.

Motive - a non-constructive proof

Theorem

Even numbers exist.

Motive - a non-constructive proof

Theorem

Even numbers exist.

Average constructive proof enjoyer:

$0 \equiv 0 \mod 2$. Therefore 0 is even. Therefore even numbers exist. $\square$

Motive - a non-constructive proof

Theorem

Even numbers exist.

Average constructive proof enjoyer:

$0 \equiv 0 \mod 2$. Therefore 0 is even. Therefore even numbers exist. $\square$

Non-constructive proof:

Suppose there are no even numbers. Then for any integer $x \in \mathbb{Z}$, $x \equiv 1 \mod 2$. But by definition of modulo, $2 \equiv 0 \mod 2$. Therefore $0 = 1$, a contradiction. $\square$

Motive - Why constructive mathematics?

Imagine the conversation:

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Are there two irrational numbers a, b, but a^b is rational?

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Are there two irrational numbers a, b, but a^b is rational?
- Me: Yes there is.

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Are there two irrational numbers a, b, but a^b is rational?
- Me: Yes there is.
- You: So, what is it?

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Are there two irrational numbers a, b, but a^b is rational?
- Me: Yes there is.
- You: So, what is it?
- Me: I can prove that it exists, but I don't know any specific a, b.

Motive - Why constructive mathematics?

Imagine the conversation:

- You: Are there two irrational numbers a, b, but a^b is rational?
- Me: Yes there is.
- You: So, what is it?
- Me: I can prove that it exists, but I don't know any specific a, b.
- You: ???

Motive - another non-constructive proof

Theorem

There are irrational numbers a, b where a^b is rational.

Motive - another non-constructive proof

Theorem

There are irrational numbers a, b where a^b is rational.

Average constructive proof enjoyer:

Take $a = \ldots, b = \ldots$, we are done. (??) $\square$

Motive - another non-constructive proof

Theorem

There are irrational numbers a, b where a^b is rational.

Average constructive proof enjoyer:

Take $a = \dots, b = \dots$, we are done. (??) $\square$

Non-constructive proof:

We know $\sqrt{2}$ is irrational. If $\sqrt{2}^{\sqrt{2}}$ is rational, then we can let $a = b = \sqrt{2}$. If not, then $a = \sqrt{2}^{\sqrt{2}}, b = \sqrt{2}$, we have

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$$

is rational. $\square$

Motive - Types in programming

1. High level languages need expressivity with a correctness guarantee - type systems.
    - FORTRAN: integers, floats
    - ALGOL 60: (additionally) boolean
    - PASCAL: (additionally) enums, tuples, arrays, recursively defined types

Motive - Types in programming

1. High level languages need expressivity with a correctness guarantee - type systems.
   - FORTRAN: integers, floats
   - ALGOL 60: (additionally) boolean
   - PASCAL: (additionally) enums, tuples, arrays, recursively defined types
2. What does it have to do with constructive mathematics? (Spoiler) it is constructive, since a valid type is always inhabited by objects, unlike mathematics, a true proposition is not necessarily inhabited by a witness.

Outline

Motive
- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results
- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Type Theory and Proof Theory

- Curry-Howard Isomorphism ("Proposition as Types")
  - (Curry 1934) Intuitionistic Implication Logic
  - (Curry 1958) Hilbert-styled deduction systems
  - (Howard 1969) Natural deduction
- Martin-Löf's Type Theories: MLTT71, MLTT72, MLTT73, MLTT79
- System F (Girard 1972, Reynolds 1974)

Outline

Motive
- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results
- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Key Contributions

- Inductively Typed: There are only 3 initial types.
- Popularized the proof-program correspondence.
- Influenced the development of interactive theorem provers, best popularized when Coq was used to prove the 4-color theorem.
- As with most type systems, has very few rules but is very sophisticated.

Outline

Motive

- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results

- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Proposition as Types, Proofs as Programs

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	$P \to Q$

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	$P \to Q$
Proof that $P \implies P$	

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	$P \to Q$
Proof that $P \implies P$	$id : P \to P$

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	$P \to Q$
Proof that $P \implies P$	$id : P \to P$
$\forall x \in A, B(x)$	$(f : A \to B(x)) : (\prod x : A) B(x)$

Proposition as Types, Proofs as Programs

Mathematics	Programming
Proposition	Type
Proof	Object of a Type
Propositions P, Q	Types P, Q
$P \wedge Q$	$Tuple(P, Q)$
$P \vee Q$	$Union(P, Q)$
$P \implies Q$	$P \to Q$
Proof that $P \implies P$	$id : P \to P$
$\forall x \in A, B(x)$	$(f : A \to B(x)) : (\prod x : A)B(x)$
$\exists x \in A, B(x)$	$(x : A, y : B(x)) : (\sum x : A)B(x)$

Example: Axiom of Choice as a Type

Theorem (Axiom of Choice)

For any nonempty collection of sets X, we have a choice function $f : X \to \bigcup X$ such that for any set $Y \in X$, $f(Y) \in Y$.

Proof.

We write this proposition concretely:

$$(\forall x \in A)(\exists y \in B) y \in x \implies (\exists f \in A \to B)(\forall x \in A) f(x) \in x$$

or rather, as a type: let A, B, C be types.

$$\left(\prod x : A\right)\left(\sum y : B\right) C(x, y) \to \left(\sum f : A \to B\right)\left(\prod x : A\right) C(x, f(x))$$

Goal: $(\prod x : A)(\sum y : B)C(x, y) \rightarrow (\sum f : A \rightarrow B)(\prod x : A)C(x, f(x))$

Now all we need to do is to find a term of the type above.
We start with assuming the LHS is given: let

$$z : (\prod x : A)(\sum y : B)C(x, y), \text{ and}$$
$$x : A.$$

Then

$$z(x) : (\sum y \in B)(C(x, y)) \tag{1}$$

Goal: $(\prod x : A)(\sum y : B)C(x, y) \rightarrow$
$(\sum f : A \rightarrow B)(\prod x : A)C(x, f(x))$

$z(x) : (\sum y \in B)(C(x, y))$ can be viewed as a pair, so we can look at its left and right entries:

$$\pi_1(z(x)) : B \tag{2}$$
$$\pi_2(z(x)) : C(x, \pi_1(z(x))) \tag{3}$$

Since $x : A$ is arbitrary, we can abstract (2) into a function (a Π-type), then call it, but preserving the type:

$$(\lambda x : \pi_1(z(x)))(x) = \pi_1(z(x)) : B \tag{4}$$

Substitute (4) into (3), we have the last term $C(x, f(x))$:

$$\pi_2(z(x)) : C(x, (\lambda x : \pi_1(z(x)))(x)) \tag{5}$$

Goal: $(\prod x : A)(\sum y : B)C(x, y) \to (\sum f : A \to B)(\prod x : A)C(x, f(x))$

We have (5):

$$\pi_2(z(x)) : C(x, (\lambda x : \pi_1(z(x)))(x))$$

Abstract x in (5):

$$\lambda x : \pi_2(z(x)) : (\prod x : A)C(x, (\lambda x : \pi_1(z(x)))(x)) \tag{6}$$

Make a pair (union type):

$$(\lambda x : \pi_1(z(x)), \lambda x : \pi_2(z(x))) :$$
$$(\sum f : A \to B)(\prod x : A)C(x, (\lambda x : \pi_1(z(x)))(x)) \tag{7}$$

Goal: $(\prod x : A)(\sum y : B)C(x,y) \to (\sum f : A \to B)(\prod x : A)C(x, f(x))$

$$(\lambda x : \pi_1(z(x)), \lambda x : \pi_2(z(x))) : (\sum f : A \to B)(\prod x : A)C(x, (\lambda x : \pi_1(z(x)))(x)) \tag{7}$$

Finally recall $z : LHS$, and we abstract it.

$$\lambda z : (\lambda x : \pi_1(z(x)), \lambda x : \pi_2(z(x))) : (\prod x : A)(\sum y : B)C(x,y) \to (\sum f : A \to B)(\prod x : A)C(x, f(x)) \tag{8}$$

The function in (8) has our desired type, therefore it is a proof of the "axiom" of choice. □

Outline

Motive

- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results

- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Discussion, Limitation, Future Work

- We have term depend on term: $f(a) = b$, type depend on term $x : A, B(x)$, but what about term depend on type? Type depend on Type?

Discussion, Limitation, Future Work

- We have term depend on term: $f(a) = b$, type depend on term $x : A, B(x)$, but what about term depend on type? Type depend on Type?
- Higher-order Type Theories, such as Calculus of Constructions (CoC) is commonly used in proof assistants.

Discussion, Limitation, Future Work

- We have term depend on term: $f(a) = b$, type depend on term $x : A, B(x)$, but what about term depend on type? Type depend on Type?
- Higher-order Type Theories, such as Calculus of Constructions (CoC) is commonly used in proof assistants.
- Two Types are equal if they have exactly the same set of proofs. When are two proofs equal?

Discussion, Limitation, Future Work

- We have term depend on term: $f(a) = b$, type depend on term $x : A, B(x)$, but what about term depend on type? Type depend on Type?
- Higher-order Type Theories, such as Calculus of Constructions (CoC) is commonly used in proof assistants.
- Two Types are equal if they have exactly the same set of proofs. When are two proofs equal? Homotopy Type Theory (Voevodsky 2005).

Outline

Motive
- Why constructive mathematics?
- Type systems

Related Work

Key Contributions

Results
- Intuitionistic Type Theory
- Axiom of Choice

Discussion

Conclusion

Conclusion

Iris

LEAN Agda