CONSTRUCTIVE LOGIC

Yinsen (Tesla) Zhang

Math Club of Penn State Harrisburg

ABOUT THE SPEAKER

'Tesla Ice Zhang'

- I work with Programming Languages and their Type Systems.
- Blog: https://ice1000.org
- GitHub: ice1000

- Second year undergraduate
- Computer Science (?) major
- Functional programming advocate

CONSTRUCTIVE

- Starting from the 20th century
- Not mainstream, but still notably popular

- It’s rebellious, it’s cool

OUTLINE

- Introduce constructive logic
  - What's bad about it?
  - What's good about it?
- Lambda calculus: your lovely proofs
  - Curry-Howard Correspondence
- Case studies
- From Set theory towards Type theory
- Extensionality

LOGICAL NOTATIONS

- Inference rules: $\frac{\text{premises}}{\text{conclusion}}$, example:

$$\frac{p \to q \quad p}{q}$$

LOGICAL NOTATIONS

- Inference rules: $\frac{\text{premises}}{\text{conclusion}}$, example:

$$\frac{p \to q \quad p}{q}$$

- Implication is right-associative
  - $p \to q \to r$ means $p \to (q \to r)$

CLASSICAL LOGIC

- In classical logic, you can prove something to be true even if you can’t construct it

CLASSICAL LOGIC

- In classical logic, you can prove something to be true even if you can't construct it
- Example: $\neg(\forall x \in A, x \leq 0) \rightarrow \exists y \in A. y > 0$

CLASSICAL LOGIC

- In classical logic, you can prove something to be true even if you can't construct it
- Example: $\neg(\forall x \in A, x \leq 0) \rightarrow \exists y \in A. y > 0$
- Example: definition of irrational numbers – numbers that are not representable by the quotient of two integers are irrational numbers

WHAT IS CONSTRUCTIVE

- The proposition or data that the propositions are on should have:
  - Formation rule
  - Construction (introduction) rule
  - Destruction (elimination) rule
- All of these should be defined, thus we can:
  - Construct a proposition
  - Construct a proof by defined introduction forms
  - Destruct a lemma by analyzing how it was constructed by case-analysis

CLASSICAL LOGIC

- In classical logic, you can prove something to be true even if you can't construct it
- Generalized proposition:

$$\neg\big(\forall x \in A, p(x)\big) \rightarrow \exists y \in A, \neg p(x)$$

- Proof-by-negation/law of exclude middle/axiom of choice?

CLASSICAL PROPOSITIONS

- Proposition. $\sqrt{2}$ is an irrational number.

- What is an irrational number? How was it defined?
  - Is it defined by formation/construction/destruction?

CONSTRUCTIVE PROPOSITIONS

- ~~Proposition. $\sqrt{2}$ is an irrational number.~~
- Proposition. $\forall n \in \mathbb{N}, m \in \mathbb{N}^+, n^2 > 2m^2 \vee n^2 < 2m^2$
- Or in other words,

$$\forall m, n \in \mathbb{N}, n^2 \equiv 2m^2 \rightarrow m \equiv 0$$

- Proof. Omitted (by doing case analysis on n and m).

PROOFS

- Proposition. It is possible that an irrational to the power of another irrational is a rational.

$$\exists a, b \in (\mathbb{R} \backslash \mathbb{Q}), a^b \in \mathbb{Q}$$

CLASSICAL PROOFS

- Proposition. It is possible that an irrational to the power of another irrational is a rational.

$$\exists a, b \in (\mathbb{R} \backslash \mathbb{Q}), a^b \in \mathbb{Q}$$

- Classical proof. Discussing the rationality of $\sqrt{2}^{\sqrt{2}}$:
  - Note: $2, \sqrt{2}^2, \sqrt{2} \times \sqrt{2}, \sqrt{2}^{\sqrt{2} \times \sqrt{2}}, \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}}$ are trivially rational as they all simplify to 2, which is known to be rational.

CONSTRUCTIVE PROOFS

- Proposition. It is possible that an irrational to the power of another irrational is a rational.

$$\exists a, b \in (\mathbb{R} \backslash \mathbb{Q}), a^b \in \mathbb{Q}$$

- Constructive proof. Give a concrete irrational number expression that is stated to exist in the proposition.
  - Proof of $\log_2 9$ is irrational is omitted
  - $\sqrt{2}^{\log_2 9} = 3.$

ATTACKS

Taking the principle of excluded middle from the mathematician would be the same, say, as proscribing the telescope to the astronomer or to the boxer the use of his fists. To prohibit existence statements and the principle of excluded middle is tantamount to relinquishing the science of mathematics altogether. For compared with the immense expanse of modern mathematics, what would the wretched remnants mean, the few isolated results, incomplete and unrelated, that the intuitionists have obtained without the use of logical ∈-axiom?

– David Hilbert, 1927

A LESS KNOWN POINT

The thrust of Bishop's work was that both Hilbert and Brouwer had been wrong about an important point on which they had agreed. Namely both of them thought that if one took constructive mathematics seriously, it would be necessary to "give up" the most important parts of modern mathematics (such as, for example, measure theory or complex analysis). Bishop showed that this was simply false, and in addition that it is not necessary to introduce unusual assumptions that appear contradictory to the uninitiated. The perceived conflict between power and security was illusory! One only had to proceed with a certain grace, instead of with Hilbert's "boxer's fists".

– Michael Beeson

LAMBDA CALCULUS

Your proofs

CONCEPTS

- “names” and “expressions”
- Like your functions’ “variables”, they are named uniquely

- Notation: x, y for names, a, b for more complicated expressions

“MORE COMPLICATED EXPRESSIONS”?

- Abstraction:

$$\lambda x.\, a$$

- Application:

$$a\ b$$

- You can wrap your expressions with parentheses

ABSTRACTIONS

- Abstraction is a term:

$$\lambda x.\, a$$

- Functions, but unnamed
- The a is a term containing usages of the variable x.

ABSTRACTIONS

- Abstraction:

$$\lambda x.\, a$$

- If we assign a name to the term $f = \lambda x.\, a$, it'll be effectively the same as defining it a math-style $f(x) = a$, where a is an expression containing variable x.
- Our lambda-calculus syntax allows us to simply strip the name off so we can combine/nest as many functions as we want

ABSTRACTIONS

- Functions with multiple parameters:

$$\lambda x.\, \lambda y.\, a$$

- Where:
  - The term $\lambda y.\, a$ can contain usages of variable x
  - The term a can contain usages of a variable y, as well as x

APPLICATIONS

- Application:

$$a\ b$$

- Term a must be an abstraction, or *evaluates* to an abstraction

APPLICATIONS

- Application when a is an abstraction:

$$(\lambda x.\, a)\; b$$

- We can *compute* this term, like we compute $1 + 1$ to 2

APPLICATIONS

- Application when a is an abstraction:

$$(\lambda x.\, a)\; b$$

- We can *compute* this term, like we compute $1 + 1$ to 2
- As a is a term containing x, we compute this term by replacing all occurrences of x in a with b

EXTENDING LAMBDA CALCULUS

- The slogan of lambda calculus is to define the basis of functions
- We can extend them with numbers and numeric operations
  - Example: $\lambda x. x^2 + 1$

EVALUATION: BY COMPARISON

- Function $f(x) = x^2 + x$
- Given arbitrary value b
- We apply it to f: $f(b)$
- It's evaluated to $b^2 + b$

- Lambda calculus term $\left(\lambda x.\, x^2 + x\right)\, b$
- Replacing all x in $x^2 + x$
- Evaluates to $b^2 + b$

CONCLUSION

- Lambda calculus defines operations on *anonymous functions*

TYPES

Describe your lambda calculus terms

CLASSIFY YOUR LAMBDA TERMS

- How can I ensure my lambda calculus term is valid?
  - Things being applied must be abstractions

CLASSIFY YOUR LAMBDA TERMS

- How can I ensure my lambda calculus term is valid?
  - Things being applied must be abstractions

- Bad example: given a lambda calculus extended with real numbers,

1 2

- Is invalid – `1` is not an abstraction!

TYPES

- Types classifies terms
  - Term is “inhabited in” or “an instance of” or simply “of” some type.
- Examples.
  - Natural number type $\mathbb{N}$, instances: 0, 1, 2, 3… (will be defined formally later)
  - Boolean type with two instances $\mathbb{B}$
  - $\mathbb{R}, \mathbb{Z}, \mathbb{Q}$
  - `int, float, int[], java.lang.Object, std::vector<int>`

TYPES

- Every lambda calculus term has a type
  - Written as $a : A$ (a is an instance of type A)
  - Examples – $1 : \mathbb{N}, \frac{1}{3} : \mathbb{Q}$

ABSTRACTION'S TYPES

- Abstraction should have type too
  - They need to know the parameter's type
  - They need to show the type after its applied

- Function types are *parameterized* by two types
  - If A and B are types, then $A \rightarrow B$ is a function type

ABSTRACTION'S TYPES

- Examples.
  - Term $\lambda x.\, x + x$ has type $\mathbb{N} \to \mathbb{N}$
  - Term $\lambda x.\, x$ has type $A \to A$ for every possible type A
  - Term $\lambda x.\, x\ 1$ has type $(\mathbb{N} \to A) \to A$ for every possible type A

WELL-TYPED TERMS

- Our term variables should have types to ensure the terms constructed on top of them are also well-typed

CONTEXT

- Variables can come from abstraction *or contexts*:

$$\Gamma ::= \epsilon \mid \Gamma'; x : A$$

$$\Gamma; x : A \vdash x : A$$

TYPED LAMBDA CALCULUS

- Contexts:

$$\Gamma ::= \epsilon \mid \Gamma'; x : A$$
$$\Gamma; x : A \vdash x : A$$

- Typing judgements:

$$\frac{\Gamma; x : A \vdash a : B}{\Gamma \vdash (\lambda x.\, a) : A \to B}$$

$$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash a\ b : B}$$

PROOF

- Well-typed lambda calculus terms are always valid

$$\frac{\Gamma; x : A \vdash a : B}{\Gamma \vdash (\lambda x.\, a) : A \to B}$$

$$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash a\ b : B}$$

CONSTRUCTIVE LOGIC

After we know lambda calculus and types

CASE ANALYSIS

- Working with naturals:

$$0 : \mathbb{N}$$

$$\frac{\Gamma \vdash a : \mathbb{N}}{\Gamma \vdash s\ a : \mathbb{N}}$$

CASE ANALYSIS

- Working with naturals:

$$0 : \mathbb{N}$$

$$\frac{\Gamma \vdash a : \mathbb{N}}{\Gamma \vdash s\ a : \mathbb{N}}$$

- Let’s define an addition operation!

CASE ANALYSIS

$$0 : \mathbb{N}$$

$$\frac{\Gamma \vdash a : \mathbb{N}}{\Gamma \vdash s\ a : \mathbb{N}}$$

- Let's define an addition operation!
- $\text{plus} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$
- $\text{plus} = \lambda \begin{Bmatrix} 0 & \to \lambda x.\, x \\ s\ a & \to \lambda x.\, s\ (\text{plus}\ a\ x) \end{Bmatrix}$

IDENTITY TYPES

- Apart from the function type, there's the identity type
- From Martin-Löf type theory

IDENTITY TYPES

- Apart from the function type, there's the identity type
- From Martin-Löf type theory

$$\frac{\Gamma \vdash a, b : A}{\Gamma \vdash a \equiv_A b}$$

$$\text{refl} : (a : A) \to a \equiv_A a$$

IDENTITY TYPES

- Introduction and Elimination
  - Dependent elimination
  - Your proofs are terms, the same as your data

IDENTITY PROPERTIES

$$\text{sym} : (a\ b : A) \to a \equiv_A b \to b \equiv_A a$$

$$\text{trans} : (a\ b\ c : A) \to a \equiv_A b \to b \equiv_A c \to a \equiv_A c$$

$$\text{cong} : (f : A \to B) \to (a\ b : A) \to a \equiv_A b \to f\ a \equiv_B f\ b$$

IDENTITY PROPERTIES

$$\text{sym} : (a\ b : A) \to a \equiv_A b \to b \equiv_A a$$
$$\text{sym} = \lambda a.\, \lambda b.\, \lambda\{\text{refl}\ a \to \text{refl}\ a\}$$

$$\text{trans} : (a\ b\ c : A) \to a \equiv_A b \to b \equiv_A c \to a \equiv_A c$$
$$\text{trans} = \lambda a.\, \lambda b.\, \lambda c.\, \lambda\{\text{refl}\ a \to \lambda x.\, x\}$$

$$\text{cong} : (f : A \to B) \to (a\ b : A) \to a \equiv_A b \to f\ a \equiv_B f\ b$$
$$\text{cong} = \lambda f.\, \lambda a.\, \lambda b.\, \lambda\{\text{refl}\ a \to \text{refl}(f\ a)\}$$

CASE STUDY

- Let's prove natural addition is commutative!

$$(n\ m : \mathbb{N}) \to \text{plus}\ n\ m \equiv_{\mathbb{N}} \text{plus}\ m\ n$$

CASE STUDY

- Let’s prove exclude-middle upon the identity type on naturals!

CASE STUDY

- Let's prove exclude-middle upon the identity type on naturals!

$$\frac{\Gamma \vdash A, B\ \mathbf{type}}{\Gamma \vdash A \vee B\ \mathbf{type}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathrm{left}\ a : A \vee B}, \frac{\Gamma \vdash b : B}{\Gamma \vdash \mathrm{right}\ b : A \vee B}$$

TYPE THEORY

- Functions are equations (algorithm/procedure)
  - Lambda calculus terms annotated with function types instead of mappings from sets to sets
- Function types are implications

CURRY-HOWARD ISOMORPHISM

- Types are Propositions, terms are proofs
- Term model

OUR FRIEND, COMPUTERS

- Computer cannot think abstractly
  - Computers are all about computation
- Imagine you have functions defined as “relation on sets”
  - How can you store a function between infinite sets?

OUR FRIEND, COMPUTERS

- Computer cannot think abstractly
  - Computers are all about computation
- Questions being asked when you want to formalize something
  - How do we compare proofs?
  - How do we read the proofs produced by computers?
  - How do we verify (double-check) the proofs?

(there are still computer proving systems based on classical set theory (ZFC), but they are the minority)

OUR FRIEND, COMPUTERS

- Computer can help compute your terms
  - Proofs based on computation are free
- You always need to define the computation rule
  - Recall that we didn't define how to compute the law of exclude middle
  - We find ways to define computation on proofs
    - Maybe someday we can, define those lost rules?

EXTENSIONALITY

- When does two type equal?
  - In set theory, we have extensionality: sets are equal if they are isomorphic
  - $\forall A, B : U. (\forall x \in A \Leftrightarrow x \in B) \Rightarrow (A \equiv B)$
  - How can we state this in type theory?

EXTENSIONALITY

- When does two type equal?
  - In set theory, we have extensionality: sets are equal if they are isomorphic
  - $\forall A, B : U.\ (\forall x \in A \Leftrightarrow x \in B) \Rightarrow (A \equiv B)$
  - How can we state this in type theory?
- Univalence axiom

$$\forall A, B : \boldsymbol{U}, A \cong B \to A \equiv_U B$$

- Where:

$$\left\{ \begin{array}{c} f : A \to B, g : B \to A \\ f \circ g \equiv_{B \to B} \lambda x. x \\ g \circ f \equiv_{A \to A} \lambda x. x \end{array} \right\} : A \cong B$$

MY RESEARCH

- Cubical type theory: constructive interpretations of the univalence axiom
- Simplicial set model can be improved
- How do we compute cubical terms
  - Anders' problem
  - Brunerie's number
- A better surface syntax, or a structural editor

ASK ME ANYTHING

Good presenters are open to audience’ doubts