Less naive type theory

Agnieszka Kozubek

Institute of Informatics
Warsaw University

26 May 2007

Plan

Lambda calculus

- origins in 1930s (Church, Curry)
- alternative foundation of mathematics
- model of computation
- basic notion: function
- function understood in an intensional way

Syntax of lambda calculus

Lambda term

- variables x, y, $\ldots$ are lambda terms
- if M, N are lambda terms then MN is also a lambda term
- if x is a variable and M is a lambda term then $\lambda x.M$ is a lambda term

Intuitive meaning

Lambda term represents a function.

- variables x, y, $\ldots$ represent some functions
- MN is an application of a function M to an argument N
- $\lambda x.M$ is a function with a parameter x and a definition M

Syntax of lambda calculus

Lambda term

- variables $x, y, \ldots$ are lambda terms
- if M, N are lambda terms then MN is also a lambda term
- if x is a variable and M is a lambda term then $\lambda x.M$ is a lambda term

Intuitive meaning

Lambda term represents a function.

- variables $x, y, \ldots$ represent some functions
- MN is an application of a function M to an argument N
- $\lambda x.M$ is a function with a parameter x and a definition M

Problem in lambda calculus

Problem

We compose arbitrary lambda terms (arbitrary functions).

Consider a function

$$f(x) = x^2$$

Problem in lambda calculus

Problem

We compose arbitrary lambda terms (arbitrary functions).

Consider a function

$$f(x) = x^2$$

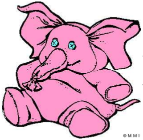
Problem in lambda calculus

Problem

We compose arbitrary lambda terms (arbitrary functions).

Consider a function

$$f(x) = x^2$$

What is $f($  $)$?

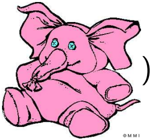
Problem in lambda calculus

Problem

We compose arbitrary lambda terms (arbitrary functions).

Consider a function

$$f(x) = x^2$$

What is $f($  $)$?

Solution

Lambda calculus with types

Type assignment systems

$$\Gamma \vdash M : \tau$$

Environment
assigns types to variables

Lambda term

Type
object describing domain and codomain of a function

Typing rules

Rules telling how to type lambda terms.

Type assignment systems

$$\Gamma \vdash M : \tau$$

Environment
assigns types to variables

Lambda term

Type
object describing domain and codomain of a function

Typing rules

Rules telling how to type lambda terms.

Type assignment systems

$$\Gamma \vdash M : \tau$$

Environment
assigns types to variables

Lambda term

Type
object describing domain and codomain of a function

Typing rules

Rules telling how to type lambda terms.

Important issue

Inhabitation problem

Is there a lambda term M such that $\vdash M : \tau$?

$$\vdash \; ? \; : \tau$$

Important issue

Inhabitation problem

Is there a lambda term M such that $\vdash M : \tau$?

$$\vdash \mathrm{M} : \tau$$

Inhabitant

Inhabited type

Important issue

Inhabitation problem

Is there a lambda term M such that $\vdash M : \tau$?

$$\vdash \mathrm{M} : \tau$$

Inhabitant

Inhabited type

Basic type system

Simply typed lambda calculus $\lambda_{\rightarrow}$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \rightarrow \sigma} \qquad \frac{\Gamma \vdash M : \tau \rightarrow \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}$$

Basic type system

Simply typed lambda calculus $\lambda_{\rightarrow}$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \rightarrow \sigma} \qquad \frac{\Gamma \vdash M : \tau \rightarrow \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}$$

Basic type system

Simply typed lambda calculus $\lambda_{\rightarrow}$

$$\Gamma, \tau \vdash \tau$$

$$\frac{\Gamma, \tau \vdash \sigma}{\Gamma \vdash \tau \rightarrow \sigma} \qquad \frac{\Gamma \vdash \tau \rightarrow \sigma \qquad \Gamma \vdash \tau}{\Gamma \vdash \sigma}$$

Basic type system

Simply typed lambda calculus $\lambda_\rightarrow$

$$\Gamma, \tau \vdash \tau$$

$$\frac{\Gamma, \tau \vdash \sigma}{\Gamma \vdash \tau \rightarrow \sigma} \qquad \frac{\Gamma \vdash \tau \rightarrow \sigma \qquad \Gamma \vdash \tau}{\Gamma \vdash \sigma}$$

Looks familiar?

Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash ? : \tau$	$\vdash \tau$?
inhabited types	provable formulas

Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash ? : \tau$	$\vdash \tau$?
inhabited types	provable formulas

Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash\ ?:\tau$	$\vdash\tau\ ?$
inhabited types	provable formulas

Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash ? : \tau$	$\vdash \tau$?
inhabited types	provable formulas

Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash\ ?:\tau$	$\vdash \tau\ ?$
inhabited types	provable formulas

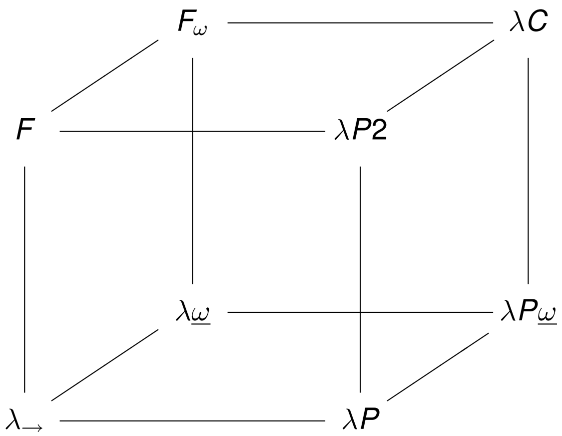
Curry-Howard isomorphism

Lambda calculus	Logic
simply typed lambda calculus $\lambda_{\rightarrow}$	minimal propositional (intuitionistic) logic
types	formulas
typing rules	proof rules
inhabitation	provability
$\vdash ? : \tau$	$\vdash \tau$?
inhabited types	provable formulas

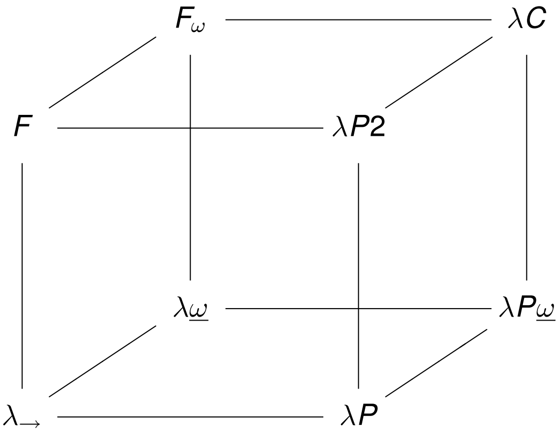
Extensions of simple types

- system λP – first-order logic
- system F – second-order propositional logic
- system F_ω – higher-order propositional logic
- calculus of constructions λC – higher-order predicate logic
- other

The λ-cube



The λ-cube



Pure Type Systems

Pure Type Systems (PTSs)

- formalism to talk about type systems
- parametric representation
- useful for comparing type systems
- beyond the cube
- used for defining new type systems (new logics)

Threat with PTSs

Example PTS – system NTT

$$\mathcal{S} = *, \Box$$
$$\mathcal{A} = * : \Box$$
$$\mathcal{R} = (*, *), (*, \Box), (*, \Box, *)$$

Inconsistent type system

Every type is inhabited $\Rightarrow$ everything is provable.

Threat with PTSs

Example PTS – system NTT

$$\mathcal{S} = *, \Box$$
$$\mathcal{A} = * : \Box$$
$$\mathcal{R} = (*, *), (*, \Box), (*, \Box, *)$$

Inconsistent type system

Every type is inhabited $\Rightarrow$ everything is provable.

System LNTT

System LNTT

$$\mathcal{S} = *^t, *^p, \Box^t, \Box^p$$

$$\mathcal{A} = *^t : \Box^t, *^p : \Box^p$$

$$\mathcal{R} = (*^t, *^t), (*^p, *^p), (*^t, *^p), (*^t, \Box^p, *^t), (*^t, \Box^t)$$

Theorem

System LNTT is consistent.

System LNTT

System LNTT

$$\mathcal{S} = *^t, *^p, \Box^t, \Box^p$$
$$\mathcal{A} = *^t : \Box^t, *^p : \Box^p$$
$$\mathcal{R} = (*^t, *^t), (*^p, *^p), (*^t, *^p), (*^t, \Box^p, *^t), (*^t, \Box^t)$$

Theorem

System LNTT is consistent.