simply typed λ-calculus

logical verification

week 2

2004 09 15

prime number theorem formalized

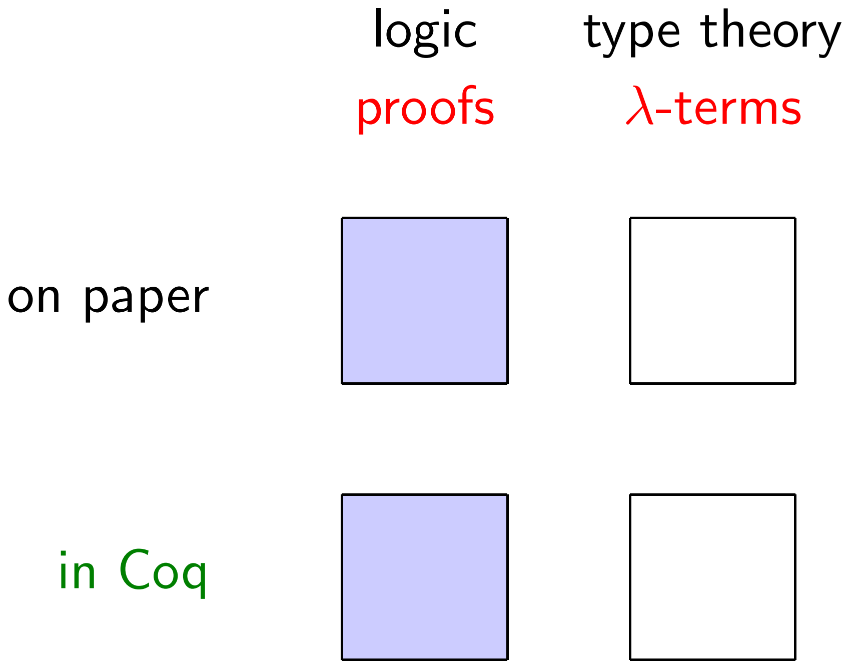
write $\pi(n)$ for the number of primes below n, then

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln(n)} = 1$$

http://www.andrew.cmu.edu/user/avigad/isabelle/

- Jeremy Avigad
- Kevin Donelly
- David Gray

last week



C program

```
#include <math.h>

double findzero(double (*f)(double), double z) {
  double x, y;
  while (x = z, y = (*f)(x), z = x - y/((*f)(x + y) - y)*y,
    fabs(z/x - 1) >= 1e-15) ;
  return z;
}

double sqrminus2(double x) { return x*x - 2; }

main() {
  printf("%.15g\n", findzero(&sqrminus2, 1));
}
```

programming styles

- imperative programming

  C

- object-oriented programming

  C++

  java

- logic programming

  prolog

- functional programming

  lisp

  ML ‘typed’

  haskell ‘lazy’ calculations with infinite data structures

functional programming

functional values become first class objects

no need to name functions anymore

```
findzero( &sqrminus2 , ...)
```

$$\downarrow$$

```
findzero( λx. x*x - 2 , ...)
```

functions also can **return** functional values

'higher order' functions

currying

$$f : A \times B \to C$$

partial evaluation

$$f(a, \cdot) : B \to C$$

curried version of the function:

$$f : A \to (B \to C)$$

$$f : A \to B \to C$$

the type of `findzero`

$$(\texttt{double} \longrightarrow \texttt{double}) \times \texttt{double} \longrightarrow \texttt{double}$$

curried:

$$(\texttt{double} \longrightarrow \texttt{double}) \longrightarrow \texttt{double} \longrightarrow \texttt{double}$$

$\uparrow$ atomic type

$\uparrow$ function type

types

- **atomic types**

  $A \; B \; C \; \ldots$

- **function types**

  $A \to B$

terms

- **variables**

  $x\ y\ z\ \ldots$

- **lambda abstraction**

  $\lambda x : A.\, t$

  the function that maps the variable x of type A to t

- **function application**

  $t\, u$

  the result of applying the function t to the argument u

parentheses

- function types associate to the right
- application associates to the left

these conventions are natural for curried functions:

$$f : A \to (B \to C)$$
$$(f\,a)\,b$$

$$\downarrow$$

$$f : A \to B \to C$$
$$f\,a\,b$$

simplest example

identity function on A

term	$\lambda x : A.\, x$
type	$A \to A$

example in the real numbers

term $\lambda x : \mathbb{R}.\ x^2 - 2$

type $\mathbb{R} \to \mathbb{R}$

$$(\lambda x : \mathbb{R}.\ x^2 - 2)\ 1 \quad = \quad 1^2 - 2 \quad = \quad -1$$

$$(\lambda x : \mathbb{R}.\ x^2 - 2)\ 2 \quad = \quad 2^2 - 2 \quad = \quad 2$$

$\uparrow$

β-step

bigger example

term	$\lambda x : (A \to B) \to C \to D.\, \lambda y : C.\, \lambda z : B.\, x\,(\lambda w : A.\, z)\,y$
type	$((A \to B) \to (C \to D)) \to C \to B \to D$

judgments

$$\underbrace{x_1 : A_1,\ x_2 : A_2,\ \ldots,\ x_n : A_n}_{\Gamma} \vdash t : A$$

context

list of variable declarations

the three typing rules

variable rule

$$\Gamma, x : A, \Gamma' \vdash x : A$$

x does not occur in Γ'

abstraction rule

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

application rule

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

type derivation for the example

$$\vdash \lambda x : (A \to B) \to C \to D.\, \lambda y : C.\, \lambda z : B.\, x\, (\lambda w : A.\, z)\, y : \\ ((A \to B) \to (C \to D)) \to C \to B \to D$$

recap minimal logic

- **formulas**

  propositional variables

  implication $A \to B$

- **rules**

  implication introduction

  implication elimination

recap example natural deduction

$$((A \to B) \to (C \to D)) \to C \to B \to D$$

implication introduction & the abstraction rule

$$\cfrac{\begin{matrix}[A^x] \\ \vdots \\ B\end{matrix}}{A \to B}\; I[x]{\to} \qquad\qquad \frac{\Gamma,\, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.\, t) : (A \to B)}$$

implication elimination & the application rule

$$\dfrac{\dfrac{\vdots}{A \to B} \qquad \dfrac{\vdots}{A}}{B}\ E{\to} \qquad\qquad \dfrac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

isomorphism

propositional variable	$\sim$	type variable
the connective $\rightarrow$	$\sim$	the type constructor $\rightarrow$
formula	$\sim$	type
assumption	$\sim$	variable
implication introduction	$\sim$	lambda abstraction
implication elimination	$\sim$	function application
proof	$\sim$	term
provability	$\sim$	'inhabitation'
proof checking	$\sim$	type checking

BHK-interpretation

Brouwer, Heyting, Kolmogorov

intuitionistic logic

proof of $A \to B$	$\sim$	function that maps proofs of A to proofs B
proof of $\bot$		does not exist
proof of $A \wedge B$	$\sim$	pair of a proof of A and a proof of B
proof of $A \vee B$	$\sim$	either a proof of A or a proof of B

propositions as types

$$\lambda x : A.\, x \;:\; A \to A$$

the function type $A \to A$ represents a proposition

the term $\lambda x : A.\, x$ represents a proof of that proposition

λ-terms are **proof objects**

Coq

term syntax

- `x`
- `fun x : A => t`
- `t u`

commands

- `Check`

  prints a term with its type

- `Print`

  print the term for a symbol with its type

example

```
fun x : A => x : A -> A
```

Coq as proof checker

'->' represents implication

Coq as functional programming language

'->' represents function type

proof objects

```
Lemma I : A -> A.
...
Qed.

Print I.
```

example

```
((A -> B) -> (C -> D)) -> C -> B -> D
```

this week

	logic proofs	type theory terms
on paper		
in Coq		