

logical verification lecture 2

2011 04 01

simply typed λ -calculus

Curry-Howard-de Bruijn isomorphism

overview: propositional logic

- minimal logic (ML)

$$((((A \rightarrow B) \rightarrow A) \rightarrow A) \rightarrow B) \rightarrow B \text{ (weak peirce)}$$

- ML + \perp

$$(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$$

- intuitionistic logic = ML + \perp + \vee + \wedge + \top

$$A \vee B \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$$

- classical logic: add a classical axiom

$$A \vee \neg A$$

plan

- proof rules and Coq tactics
- proof rules and typing rules
simply typed lambda-calculus
Curry-Howard isomorphism
- classical logic

apply and implication elimination

$$\frac{A \rightarrow B \quad A}{B} E \rightarrow$$

goal: B

assumption: $x : A \rightarrow B$

tactic: apply x

new goal: A

NB: apply versus assumption

NB: apply with

intro and implication introduction

$$\frac{B}{A \rightarrow B} \text{I}[x] \rightarrow$$

goal: $A \rightarrow B$

tactic: intro x

new goal: B

falsum elimination

$$\frac{\perp}{A} E_{\perp}$$

goal: A

tactic: elimtype False

new goal: \perp

split and conjunction introduction

$$\frac{A \quad B}{A \wedge B} I \wedge$$

goal: $A \wedge B$

tactic: split

new goals: A and B

left, right and disjunction introduction

$$\frac{A}{A \vee B} I\vee \quad \frac{B}{A \vee B} I\vee$$

goal: $A \vee B$

tactic: left

new goal: A

elim and conjunction elimination

$$\frac{A \wedge B}{A} E\wedge \quad \frac{A \wedge B}{B} E\wedge$$

goal: A

assumption: $x : A \wedge B$

tactic: elim x

new goal: $A \rightarrow B \rightarrow A$

(after two intros we have A and B available as hypothesis)

elim and disjunction elimination

$$\frac{A \vee B \quad A \rightarrow C \quad B \rightarrow C}{C}$$

goal: C

assumption: $x : A \vee B$

tactic: elim x

new goals: $A \rightarrow C$ and $B \rightarrow C$

from logic to type theory

	logic	type theory
on paper	proofs	λ -terms
in Coq	×	today
	×	today

programming styles

- imperative programming
C
- object-oriented programming
C++ Java
- logic programming
prolog
- functional programming
ML Haskell

simply typed λ -calculus: types

- type variable (atomic type)

a b c ...

- functional type

(A \rightarrow B)

simply typed λ -calculus: terms

- variable

x

- lambda abstraction

$(\lambda x : A. M)$

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

- function application

$(F M)$

the application of the function F to the argument M

lambda-terms: examples

term: $\lambda x : A. x$

type: $A \rightarrow A$

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

type: $\mathbb{R} \rightarrow \mathbb{R}$

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

type: $((A \rightarrow B) \rightarrow C \rightarrow D) \rightarrow C \rightarrow B \rightarrow D$

lambda-terms: more examples

$\lambda x:A. x : A \rightarrow A$

$\lambda x:A. \lambda y:B. x : A \rightarrow B \rightarrow A$

$\lambda x:A. \lambda y:A. x : A \rightarrow A \rightarrow A$

$\lambda x:A. \lambda y:A. y : A \rightarrow A \rightarrow A$

judgement

$$x_1 : A_1, \dots, x_n : A_n \vdash M : A$$

in an environment $x_1 : A_1, \dots, x_n : A_n$ the term M has type A

typing rules

variable rule:

$$\Gamma \vdash x : A \quad \text{if } x : A \in \Gamma$$

abstraction rule:

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

application rule:

$$\frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash M : A}{\Gamma \vdash (F M) : B}$$

lambda-terms: examples reconsidered

$$\frac{x : A \vdash x : A}{\vdash \lambda x : A. x : A \rightarrow A}$$

$$\frac{\frac{x : A, y : B \vdash x : A}{x : A \vdash \lambda y : B. x : B \rightarrow A}}{\vdash \lambda x : A. \lambda y : B. x : A \rightarrow B \rightarrow A}$$

Curry-Howard-De Bruijn isomorphism

Curry- Howard- De Bruijn- isomorphism:
logic \sim typed λ -calculus

today more in particular:
minimal propositional logic \sim simply typed λ -calculus

minimal logic

only the connective \rightarrow

- formulas:

propositional variables

implication $A \rightarrow B$

- logical rules:

implication introduction

implication elimination

assumption

formulas as types

propositional variable a \sim type variable a

implication $A \rightarrow B$ \sim function space $A \rightarrow B$

proofs as terms: 1

$$\frac{\begin{array}{c} [A^x] \\ \vdots \\ B \end{array}}{A \rightarrow B} \text{I}[x] \rightarrow \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A. M) : A \rightarrow B}$$

proofs as terms: 2

implication elimination and application

$$\frac{A \rightarrow B \quad A}{B} E \rightarrow \quad \frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash M : A}{\Gamma \vdash (F M) : B}$$

Curry-Howard-De Bruijn isomorphism

formula \sim type

propositional variable \sim type variable

connective \rightarrow \sim type constructor \rightarrow

proof \sim term

assumption \sim term variable

implication introduction \sim abstraction

implication elimination \sim application

example: proof objects

- $\lambda x : A. x : A \rightarrow A$
the function type $A \rightarrow A$ represents a proposition
the term $\lambda x : A. x$ represents a proof of that proposition
- $\lambda x : A. \lambda y : B. x : A \rightarrow B \rightarrow A$
- $\lambda x : A. \lambda y : A. x : A \rightarrow A \rightarrow A$
- $\lambda x : A. \lambda y : A. y : A \rightarrow A \rightarrow A$

alpha equivalence

we identify expressions
that are equal
up to a renaming of bound variables

alpha equivalence: examples

same terms:

$\lambda x : A. x$ and $\lambda y : A. y$

same proofs:

$\frac{[A^x]}{A \rightarrow A} \text{I}[x] \rightarrow$

and

$\frac{[A^y]}{A \rightarrow A} \text{I}[y] \rightarrow$

alpha equivalence: non-examples

different terms:

$\lambda x : A. \lambda y : A. x$ and $\lambda x : A. \lambda y : A. y$

different proofs:

$$\frac{\frac{[A^x]}{A \rightarrow A} \quad I[y] \rightarrow}{A \rightarrow A \rightarrow A} \quad I[x] \rightarrow$$

and

$$\frac{\frac{[A^y]}{A \rightarrow A} \quad I[y] \rightarrow}{A \rightarrow A \rightarrow A} \quad I[x] \rightarrow$$

Curry-Howard-De Bruijn isomorphism

type checking:

is the typing derivation $\Gamma \vdash P : A$ correct?

corresponds to

proof checking

is the proof P of the formula A using assumptions Γ correct?

and is decidable for λ^{\rightarrow} and ML

Curry-Howard-De Bruijn isomorphism

inhabitation

do we have a closed term P of type A ?

corresponds to

provability

do we have a proof P of the formula A ?

decidable for λ^{\rightarrow} and ML

(but for instance not for first-order predicate logic)

Coq term syntax

x
 $fun\ x : A \Rightarrow M$
 $M\ N$

x
 $\lambda x : A. M$
 $M\ N$

Coq commands

- Check
prints a term with its type
- Print
prints the term for a symbol with its type
- Definition
binds a term to an identifier

classical logic

start with intuitionistic logic

add a classical axiom

from intuitionistic to classical logic

- add the law of excluded middle

$$A \vee \neg A$$

- add the double negation rule

$$\neg\neg A \rightarrow A$$

- add Peirce's law

$$((A \rightarrow B) \rightarrow A) \rightarrow A$$

classical logic: examples of tautologies

assume the law of excluded middle

- double negation

$$\neg\neg A \rightarrow A$$

- Peirce's Law

$$((A \rightarrow B) \rightarrow A) \rightarrow A$$

constructive versus classical logic

constructive point of view:

main issue: provability

does the formula have a proof?

classical point of view:

main issue: truth

is the formula true?

constructive point of view (background)

proof of $A \rightarrow B$	\sim	function that maps proofs of A to proofs of B
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
proof of \perp		does not exist