

Type theory and logic

Lecture IV: meta-theoretical reasoning

4 July 2014

柯向上

Department of Computer Science
University of Oxford

Hsiang-Shang.Ko@cs.ox.ac.uk

Meta-language vs object language

Types and programs form a language, which are talked about by a separate language of judgements and derivations. In this case we call the former the *object language*, and the latter the *meta-language*.

What we write down as types and programs are nothing more than certain syntax trees by themselves; then, at a higher level, we organise and relate these syntax trees with judgements and derivations.

For example, equality judgements are a meta-theoretic notion and cannot be used inside the theory to state equations as provable propositions — we need identity types instead.

Type theory should eat itself

Judgements and derivations can also be regarded as syntax trees to be reasoned about. For example, consistency is a statement in which judgements and derivations are the object language and English is the meta-language. (Canonicity is another example.)

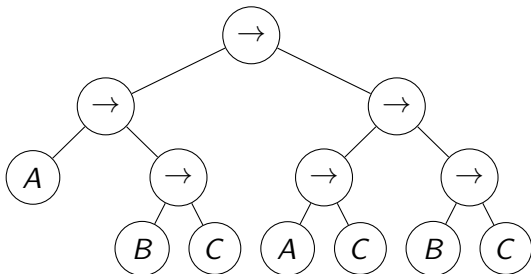
We can also use Agda as the meta-language!

Implicational fragment of propositional logic

Today we consider only propositions formed with implication.

Each of these propositions is a finite tree whose internal nodes are implications and whose leaves are atomic propositions, which are elements of a given set $Var = \{A, B, C, \dots\}$.

Example. The proposition $(A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow C) \rightarrow B \rightarrow C$ is represented as



Natural deduction

Natural deduction is the type part of simple type theory (and we are considering only the implicational fragment).

$$\frac{}{\Gamma \vdash p} \text{ (assum) } \quad \text{when } p \in \Gamma$$

$$\frac{\Gamma, p \vdash q}{\Gamma \vdash p \rightarrow q} (\rightarrow I)$$

$$\frac{\Gamma \vdash p \rightarrow q \quad \Gamma \vdash p}{\Gamma \vdash q} (\rightarrow E)$$

Untyped λ -calculus

A λ -term is either a variable, an abstraction, or an application.

We usually assume *α -equivalence* of λ -terms, i.e., the names of *bound variables* do not matter.

- Change of bound variable names is called *α -conversion*, which has to be *capture-avoiding*, i.e., *free variables* must not become bound after a name change.
- In formalisation, we prefer not to deal with α -equivalence explicitly, and one way is to use *de Bruijn indices* — λ 's are nameless, and a bound variable is represented as a natural number indicating to which λ it is bound.

Simply typed λ -calculus (à la Curry)

λ -calculus was designed to model function abstraction and application in mathematics. In untyped λ -calculus, however, we can write nonsensical terms like $\lambda x. x x$.

We can use the implicational fragment of propositional logic as a type language for λ -calculus, ruling out nonsensical terms.

$$\frac{}{\Gamma \vdash x : p} \text{ (var) } \quad \text{when } x : p \in \Gamma$$

$$\frac{\Gamma, x : p \vdash t : q}{\Gamma \vdash \lambda x. t : p \rightarrow q} \text{ (abs)}$$

$$\frac{\Gamma \vdash t : p \rightarrow q \quad \Gamma \vdash u : p}{\Gamma \vdash t u : q} \text{ (app)}$$



Haskell Curry

picture stolen from <http://bodil.org/hipster>

Curry–Howard isomorphism

Derivations in natural deduction and well-typed λ -terms are in one-to-one correspondence.

That is, we can write two functions,

- one mapping a logical derivation in natural deduction to a λ -term and its typing derivation, and
- the other mapping a λ -term with a typing derivation to a logical derivation in natural deduction,

and can prove that the two functions are inverse to each other.

This result is historically significant: two formalisms are developed separately from logical and computational perspectives, yet they coincide perfectly.

Simply typed λ -calculus à la Church

The Curry–Howard isomorphism points out that derivations in natural deduction are actually λ -terms in disguise.

These λ -terms are intrinsically typed, so every term we are able to write down is necessarily well-behaved, whereas in simply typed λ -calculus à la Curry, we can write arbitrary λ -terms, and only rule out ill-behaved ones via typing later.



Alonzo Church

picture stolen from <http://bodil.org/hipster>



Super lambda bros.

picture stolen from <http://bodil.org/hipster>



The Curry–Howard isomorphism

picture stolen from <http://bodil.org/hipster>

Semantics

After defining a language (like the implicational fragment of propositional logic), which consists of a bunch of syntax trees, we need to specify what these trees mean.

Judgements and derivations (which form a *deduction system*) assign meaning to the propositional language by specifying how it is used in formal reasoning.

We can also translate the syntax trees into entities in a well understood semantic domain. In the case of propositional logic, we can translate propositional trees to functions on truth values. (This is the classical treatment.)

Two-valued semantics of propositional logic

- Define $\text{Bool} := \{\text{false}, \text{true}\}$.
- An *assignment* is a function of type $V \rightarrow \text{Bool}$.
- A proposition p is translated into a function $\llbracket p \rrbracket : (V \rightarrow \text{Bool}) \rightarrow \text{Bool}$ mapping assignments to truth values.
- An assignment σ *models* a proposition p exactly when $\llbracket p \rrbracket \sigma$ is true, and *models* a context Γ exactly when it models every proposition in Γ .

Two-valued semantics of propositional logic

- A proposition p is *satisfiable* exactly when there exists an assignment that models p , and is *valid* exactly when every assignment models p .
- A proposition p is a *semantic consequence* of a context Γ (written $\Gamma \models p$) exactly when every assignment that models Γ also models p .

Exercise. Show that $(p \rightarrow p \rightarrow q) \rightarrow (p \rightarrow q)$ is valid for any propositions p and q .

Exercise. Show that a proposition p is valid if and only if p is a semantic consequence of the empty context.

Relationship between deduction systems and semantics

Natural deduction is *sound* with respect to the two-valued semantics: whenever we can deduce $\Gamma \vdash p$, it must be the case that $\Gamma \models p$.

The implicational fragment of propositional logic is also *(semantically) complete* with respect to the two-valued semantics: if $\Gamma \models p$, then we can construct a derivation of $\Gamma \vdash p$.