Propositions as Types
Curry-Howard Isomorphism

Klaus Ostermann
Aarhus University

Curry-Howard Isomorphism

- There is a deep connection between type systems and (intuitionstic/constructive) logic
- A proof of a proposition in constructive logic is a construction of an object that witnesses the proposition
- The Curry-Howard isomorphism says that proofs are the same as terms/programs

Constructive vs classical proofs

- Not every proof in classical logic is also valid in intuitionistic logic

Theorem There exist irrational numbers a and b such that a^b is rational.

Proof. Either $\sqrt{2}^{\sqrt{2}}$ is rational or not. If it is, take $a = b = \sqrt{2}$ and we are done. If it is not, take $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$; then $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$, and again we are done. $\square$

- Law of excluded middle is not valid in intuitionistic logic: It is not constructive!

Intuitionistic logic

- Syntax of formulas:

$$\phi \quad ::= \quad \top \mid \bot \mid P \mid \phi_1 \Rightarrow \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi.$$

- With second-order quantification:

$$\phi \quad ::= \quad \cdots \mid \forall P\,.\,\phi.$$

Natural Deduction

- Calculus developed by Gentzen to define proof rules of a logic
- Operators (so-called connectives) typically have introduction and elimination rules
- We will see that the deduction rules in natural deduction style correspond exactly to the typing rules of System F with sums and products
  - Terms are a linear notation of proofs!

Proof- and Typing Rules Side-by-Side

	intuitionistic logic	$\lambda^{\rightarrow}$ *or System F type system*
(axiom)	$\Gamma, \phi \vdash \phi$	$\Gamma, x : \tau \vdash x : \tau$
(→-intro)	$\dfrac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \Rightarrow \psi}$	$\dfrac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x : \sigma . e) : \sigma \rightarrow \tau}$
(→-elim)	$\dfrac{\Gamma \vdash \phi_1 \Rightarrow \phi_2 \quad \Gamma \vdash \phi_1}{\Gamma \vdash \phi_2}$	$\dfrac{\Gamma \vdash e_0 : \sigma \rightarrow \tau \quad \Gamma \vdash e_1 : \sigma}{\Gamma \vdash (e_0\ e_1) : \tau}$
(∧-intro)	$\dfrac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$	$\dfrac{\Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1, e_2) : \sigma * \tau}$
(∧-elim)	$\dfrac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \quad \dfrac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$	$\dfrac{\Gamma \vdash e : \sigma * \tau}{\Gamma \vdash \#1\, e : \sigma} \quad \dfrac{\Gamma \vdash e : \sigma * \tau}{\Gamma \vdash \#2\, e : \tau}$

Proof- and Typing Rules Side-by-Side

	intuitionistic logic	$\lambda^{\rightarrow}$ *or System F type system*
($\vee$-intro)	$\dfrac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}$ $\quad$ $\dfrac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$	$\dfrac{\Gamma \vdash e : \sigma}{\Gamma \vdash \mathbf{inl}_{\sigma+\tau} : e\sigma + \tau}$ $\quad$ $\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{inr}_{\sigma+\tau} : e\sigma + \tau}$
($\vee$-elim)	$\dfrac{\Gamma \vdash \phi \vee \psi \quad \Gamma \vdash \phi \rightarrow \chi \quad \Gamma \vdash \psi \rightarrow \chi}{\Gamma \vdash \chi}$	$\dfrac{\Gamma \vdash e : \sigma + \tau \quad \Gamma \vdash e_1 : \sigma \rightarrow \rho \quad \Gamma \vdash e_2 : \tau \rightarrow \rho}{\Gamma \vdash \mathbf{case}\ e_0\ \mathbf{of}\ e_1 \mid e_2 : \rho}$
($\forall$-intro)	$\dfrac{\Gamma, P \vdash \phi}{\Gamma \vdash \forall P . \phi}$	$\dfrac{\Delta, \alpha; \Gamma \vdash e : \tau \quad \alpha \notin FV(\Gamma)}{\Delta; \Gamma \vdash (\Lambda\alpha. e) : \forall\alpha . \tau}$
($\forall$-elim)	$\dfrac{\Gamma \vdash \forall P . \phi}{\Gamma \vdash \phi\{\psi/P\}}$	$\dfrac{\Delta; \Gamma \vdash e : \forall\alpha . \tau \quad \Delta \vdash \sigma}{\Delta; \Gamma \vdash (e\ \sigma) : \tau\{\sigma/\alpha\}}$

The Curry-Howard Isomorphism

- A.k.a as "Propositions as Types"

	type theory		*logic*
τ	type	ϕ	proposition
τ	inhabited type	ϕ	theorem
e	well-typed program	π	proof
$\rightarrow$	function space	$\rightarrow$	implication
$*$	product	$\wedge$	conjunction
$+$	sum	$\vee$	disjunction
$\forall$	type quantifier	$\forall$	2nd order quantifier
B	inhabited type	$\top$	truth
void	uninhabited type	$\perp$	falsity

Logical Interpretation of Program Transformations

- Reduction = Proof Normalization
  - Existence of normal form can be formalized as *Cut Elimination Theorem* (Gentzen's "Hauptsatz")
  - Typically presented using sequent calculus rather than natural deduction
- Curry and uncurry are proofs of

$$\forall P, Q, R \, . \, (P \wedge Q \rightarrow R) \;\leftrightarrow\; (P \rightarrow Q \rightarrow R)$$

- CPS Transformation relates intuitionistic to classical logic

Inconsistent type systems

- Many practical type systems are inconsistent when viewed as a logic
- For example, a fixed-point operator **`fix : ∀a. (a->a) -> a`** makes the type system inconsistent, because (fix id) has type ∀a.a, i.e., every type is inhabited
- In Haskell/ML-like languages, the CH-Isomorphism holds „modulo termination"
- Hard to apply to object-oriented type systems (nominal type systems, null pointers etc. all make it more difficult to view them through the lense of CH)

Towards theorem proving

- Quantification in System F is over propositions
- To quantify over objects dependent types are needed
- Dependent types are types that are parameterized by values. The binder is often called ∀ or Π

$$\frac{\Gamma \vdash S :: * \qquad \Gamma, x{:}S \vdash t : T}{\Gamma \vdash \lambda x{:}S.t : \Pi x{:}S.T} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : \Pi x{:}S.T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1\, t_2 : [x \mapsto t_2]T} \qquad \text{(T-APP)}$$

- Many theorem provers are based on dependent type theory
  - Coq, Twelf, …
- [You don‘t need to understand dependent types the exam]