

# 2010 Summer School on Dependable Software

Zhong Shao

Yale University

*Lecture 2*  
*Curry-Howard Isomorphism*  
*July 20, 2010*

# Outline

*Learn how to reason formally!*

Topics:

- Propositional and predicate logics
- Typed lambda calculi
- Curry-Howard isomorphism
- Inductive definitions

# Part I: Propositional Logic

- Propositional logic
  - Propositions (formulas / declarative sentences)
  - Sequents (judgements)
  - Inference rules and axioms
- How to prove ?
  - Hilbert presentation
  - Natural deduction
  - Natural deduction in sequent style
  - Sequent calculus
- Properties of a proof system
  - Soundness
  - Completeness
  - Consistency

# Propositional language

*A propositional language L contains*

- a set of atomic sentences:

$\{p, q, r, \dots \dots \}$

- a set of operators for putting them together:

and	$\wedge$	(arity 2)
or	$\vee$	(arity 2)
implies	$\rightarrow$	(arity 2)
not	$\neg$	(arity 1)
falsity	$\perp$	(arity 0)

# Proof-based view of logic

Given a set of "assumptions" (or premises), we want to prove a "conclusion":

$$\boxed{\Gamma \vdash A} \quad \text{where } \Gamma = A_1, A_2, \dots, A_n$$

This is called a "sequent"

The game of "logic" is to design a very small set of rules (and axioms) to do the "reasoning".

A proof is a syntactic demonstration that a conclusion follows from a set of premises.

# Proof-based view of logic

Some of the most common systems:

- Hilbert-Frege presentations
  - Pros: concise and small set of axioms and rules; good for proving meta-properties about the logic
  - Cons: axioms don't match intuition; hard to build proof
- Natural deduction
  - No axioms; only inference rules that match well our intuition about each logic operator
- Natural deduction in sequent style
  - Assumption discharge is made more explicit
- Sequent calculus
  - All inference rules satisfy subformula properties.

# Hilbert-Frege presentations

*Under Hilbert-Frege, the propositional classic logic can be characterized using the following axioms and inference rule:*

**Three axioms:**

$$\text{Ax1} \quad A \rightarrow (B \rightarrow A)$$

$$\text{Ax2} \quad (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$$

$$\text{Ax3} \quad (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$$

**One inference rule (a.k.a. *modus ponens*):**

$$\text{MP} \quad \frac{A \quad A \rightarrow B}{B}$$

# Hilbert-Frege presentations

The operators  $\wedge$ ,  $\vee$ , and  $\perp$  are derived forms:

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

$$A \vee B \equiv \neg A \rightarrow B$$

$$\perp \equiv \neg(A \rightarrow A)$$

A proof of the sequent  $\Gamma \vdash A$  (i.e., showing  $A$  from the premises  $\Gamma$ ) is a finite sequence of formulas ending with  $A$  such that each formula is either

- one of the axioms, or
- a member of  $\Gamma$ , or
- derivable from previous formulas using MP

# Hilbert-Frege presentations

Hilbert-Frege proof for	$\emptyset \vdash \neg q \rightarrow (q \rightarrow p)$	
1.	$\neg q \rightarrow (\neg p \rightarrow \neg q)$	Ax1
2.	$(\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$	Ax3
3.	$((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)) \rightarrow$ $(\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)))$	Ax1
4.	$\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p))$	MP 2,3
5.	$(\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p))) \rightarrow$ $((\neg q \rightarrow (\neg p \rightarrow \neg q)) \rightarrow (\neg q \rightarrow (q \rightarrow p)))$	Ax2
6.	$(\neg q \rightarrow (\neg p \rightarrow \neg q)) \rightarrow (\neg q \rightarrow (q \rightarrow p))$	MP 4,5
7.	$\neg q \rightarrow (q \rightarrow p)$	MP 1,6

Problems: the axioms don't match our intuition about  $\neg$  and  $\rightarrow$  --- proofs are harder to find.

# Natural deduction

Invented by G. Gentzen in 1934. No axioms, only rules of inference.

At each stage of a proof, can introduce any formula as hypothesis (which can be discharged later by other rules).

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ B \end{array}}}{A \rightarrow B} \rightarrow i \qquad \frac{A \quad A \rightarrow B}{B} \rightarrow e$$

# Natural deduction

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

$$\frac{A \wedge B}{A} \wedge e_1 \quad \frac{A \wedge B}{B} \wedge e_2$$

$$\frac{A}{A \vee B} \vee i_1 \quad \frac{B}{A \vee B} \vee i_2$$

$$\frac{A \vee B \quad \boxed{\begin{array}{c} A \\ \vdots \\ C \end{array}} \quad \boxed{\begin{array}{c} B \\ \vdots \\ C \end{array}}}{C} \vee e$$

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ \perp \end{array}}}{\neg A} \neg i$$

$$\frac{A \quad \neg A}{\perp} \neg e$$

no intro rule for  $\perp$

$$\frac{\perp}{A} \perp e$$

# Natural deduction

The previous set of rules define “propositional intuitionistic logic”

With the following rule, it becomes “propositional classic logic”

$$\frac{\neg\neg A}{A} \neg\neg$$

This allows you to “prove by contradiction”

# Natural deduction

Natural deduction proof for  $\emptyset \vdash \neg q \rightarrow (q \rightarrow p)$

1.	$\neg q$	Assumption
2.	$q$	Assumption
3.	$\perp$	$\neg e$ 1,2
4.	$p$	$\perp e$ 3
5.	$q \rightarrow p$	$\rightarrow i$ 2-4
6.	$\neg q \rightarrow (q \rightarrow p)$	$\rightarrow i$ 1-5

# Natural deduction in sequent style

Goal: to make discharging of assumptions clear

Deal with "sequents" at each step

A proof is a tree, whose root is the sequent being proved and whose leaves are "basic sequents" of the form:

$$A \vdash A$$

# Natural deduction in sequent style

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \wedge_i \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{e1} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{e2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{i1}$$

$$\frac{\Gamma \vdash A \vee B \quad \Delta, A \vdash C \quad \Theta, B \vdash C}{\Gamma, \Delta, \Theta \vdash C} \vee_e$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{i2}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_i$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B} \rightarrow_e$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg_i$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash \neg A}{\Gamma, \Delta \vdash \perp} \neg_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_e$$

# Natural deduction in sequent style

A few variations:

- More general form of basic sequents:

$$\Gamma, A \vdash A$$

- Replace rules w. more than one premise by:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge i$$

- Add rule of monotonicity:

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}$$

# Properties of propositional logic

Question: how do we tell whether the set of inference rules really match what we want?

What is the "semantics" of propositional logic?

- Define the set of truth values: t and f
- Give the meanings of  $\wedge, \vee, \rightarrow, \neg, \perp$ .

The semantic relationship between a set of assumptions and a conclusion:

$$\Gamma \models A$$

For any truth-value assignment to atomic formulas in  $\Gamma$  and  $A$ , if each formula in  $\Gamma$  computes to "t", then  $A$  computes to "t" as well.

# Semantics of propositional logic

A propositional language  $L$  is a pair  $(P, O)$  consisting of a countable set  $P$  of atomic sentences together with a set  $O$  of operators (each operator comes with an arity).

A valuation system for a propositional language  $L$  is a triple  $(M, D, F)$  where

- $M$  is a set with at least two elements, the set of truth values
- $D$  is a non-empty proper subset of  $M$ , the set of designated truth values
- $F = (f_{o_1}, f_{o_2}, \dots, f_{o_p})$  is a set of functions, one for each operator in  $O = \{o_1, o_2, \dots, o_n\}$ , such that  $f_o : M^n \rightarrow M$  (where  $n$  is the arity of  $o$ ). We say  $f_o$  interprets  $o$ .

# Semantics of propositional logic

An assignment  $a$  under a valuation system  $(M, D, F)$  for a language  $L = (P, O)$  is a function

$$a : P \rightarrow M$$

Each assignment  $a$  for a valuation system  $(M, D, F)$  induces an interpretation  $v_a$  given by:

- $v_a(p) = a(p)$  for  $p \in P$
- $v_a(o(A_1, \dots, A_n)) = f_o(v_a(A_1), \dots, v_a(A_n))$  where  $n$  is the arity of  $o$  and  $f_o$  interprets  $M$ .

An interpretation is a valuation system plus an assignment.

# Semantics of propositional logic

Classical propositional logic:

$$M = \{t, f\}$$

$$D = \{t\}$$

$$F = \{f_{\wedge}, f_{\vee}, f_{\rightarrow}, f_{\neg}, f_{\perp}\} \text{ where}$$

$f_{\wedge}$		t	f	$f_{\vee}$		t	f	$f_{\rightarrow}$		t	f	$f_{\neg}$		$f_{\perp}$	
t		t	f	t		t	t	t		t	f	t		f	
f		f	f	f		t	f	f		t	t	f		t	

# Important properties

A presentation  $\vdash$  is **sound with respect to a semantics**  $\models$  if for all  $\Gamma$  and  $A$ , if  $\Gamma \vdash A$ , then  $\Gamma \models A$ .

A presentation  $\vdash$  is **complete with respect to a semantics**  $\models$  if for all  $\Gamma$  and  $A$ , if  $\Gamma \models A$ , then  $\Gamma \vdash A$ .

Given a presentation  $\vdash$ , a set of formulas  $\Gamma$  is inconsistent iff for some formula  $A$ , we have both  $\Gamma \vdash A$  and  $\Gamma \vdash \neg A$ .

A presentation  $\vdash$  is **negation consistent** if there is no  $A$  such that  $\emptyset \vdash A$  and  $\emptyset \vdash \neg A$ .

# Important properties

A presentation  $\vdash$  is **absolute consistency** if there exists an  $A$  such that  $\emptyset \not\vdash A$ .

For classic/intuitive logics, **negation consistency** and **absolute consistency** are equivalent.

Given a presentation  $\vdash$ , a set of formulas  $\Gamma$  is **complete** if for every closed formula  $A$ , either  $\Gamma \vdash A$  or  $\Gamma \vdash \neg A$ .

[This is not same as "**complete w. r. t. a semantics  $\models$** ".]

Inconsistent logics are always complete! But they are useless!

# Part II: Lambda Calculus

## Lambda calculus (untyped)

- Syntax
- $\beta$ -conversion; normal form
- Church-Rosser; infinite reduction (fixpoint)

## Simply typed lambda calculus (Church-style)

- Typing rules
- Strong normalization
- Subject reduction

## Curry-Howard isomorphism

# Lambda calculus

Invented by Church in 1932.

It is as powerful as Turing machine.

Syntax:

$$e ::= x \mid \lambda x.e \mid e_1 e_2$$

Basis for many functional languages  
(Scheme, Lisp, ML, Haskell)

# Lambda calculus

Syntactic sugar:

$$ee_1 \dots e_n = (\dots((ee_1)e_2)\dots e_n)$$

$$\lambda x_1 \dots x_n. e = \lambda x_1. (\lambda x_2. (\dots (\lambda x_n. e) \dots))$$

**Free and bound variables:** a variable  $x$  is bound in  $e$  iff it is in the scope of an occurrence of  $\lambda x$  in  $e$ , otherwise it is free in  $e$ .

Bound variables can be renamed via  **$\alpha$ -conversion**

$$\lambda x. e \equiv_{\alpha} \lambda y. [y/x]e \quad \text{if } y \notin FV(e)$$

# Free variables

The set of free variables in  $e$ , denoted as  $FV(e)$  is defined as:

$$FV(x) = \{x\};$$

$$FV(e_1 e_2) = FV(e_1) \cup FV(e_2);$$

$$FV(\lambda x. e) = FV(e) - \{x\}.$$

$e$  is a closed  $\lambda$ -term if  $FV(e) = \emptyset$ .

# Substitutions

The substitution  $[e'/x]e$  is defined as:

$$[e'/x] x = e'$$

$$[e'/x] y = y$$

$$[e'/x] (e_1 e_2) = ([e'/x] e_1) ([e'/x] e_2)$$

$$[e'/x] (\lambda x. e_1) = \lambda x. e_1$$

$$[e'/x] (\lambda y. e_1) = \lambda y. e_1 \quad \text{if } x \notin FV(e_1).$$

$$[e'/x] (\lambda y. e_1) = \lambda y. [e'/x] e_1 \quad \text{if } x \in FV(e_1) \text{ \& } y \notin FV(e').$$

$$[e'/x] (\lambda y. e_1) = \lambda z. [e'/x] [z/y] e_1 \quad \text{if } x \in FV(e_1) \text{ \& } y \in FV(e').$$

# $\beta$ reduction

A  $\beta$ -redex is a term of form  $(\lambda x.e)e'$ . The result  $[e'/x]e$  is called its contractum

$\beta$ -reduction:  $(\lambda x.e)e' \rightarrow_{\beta} [e'/x]e$

If  $e_1 \rightarrow_{\beta} e_2$  then  $ee_1 \rightarrow_{\beta} ee_2$ , and  $e_1e \rightarrow_{\beta} e_2e$ ,  
and  $\lambda x.e_1 \rightarrow_{\beta} \lambda x.e_2$ .

We write  $e \twoheadrightarrow_{\beta} e'$  if  $e \rightarrow_{\beta} \dots \rightarrow_{\beta} e'$

$\equiv_{\beta}$  is the equivalence relation induced from  $\twoheadrightarrow_{\beta}$

# Normal form

A  $\lambda$ -term  $e$  is a  **$\beta$ -normal form** if it does not have any  $\beta$ -redex as subexpression.

A term  $e$  has a  $\beta$ -normal form if for some  $e'$ : (1)  $e \equiv_{\beta} e'$ ; (2)  $e'$  is a  $\beta$ -normal form.

Some  $\lambda$ -terms do not have any normal form:

$$(\lambda x. xx)(\lambda x. xx) \rightarrow_{\beta} ?$$

# Some properties for lambda calculus

If  $e$  has a normal form, then this is the only one it will have (up to  $\equiv_\alpha$ )

The  $\beta$ -reduction satisfies the so-called **Church-Rosser** property: if  $e \twoheadrightarrow_\beta e_1$  and  $e \twoheadrightarrow_\beta e_2$ , then there exists  $e_3$  such that  $e_1 \twoheadrightarrow_\beta e_3$  and  $e_2 \twoheadrightarrow_\beta e_3$

A term  $e$  is called ***strongly normalizing*** iff all reduction sequences starting with  $e$  terminate.

# Simply typed lambda calculus

Motivation: can we put some restrictions on the "syntax" of  $\lambda$ -terms so that non-normalizing terms such as  $(\lambda x.xx)(\lambda x.xx)$  can be ruled out.

We are primarily interested in Church-style "simply typed  $\lambda$ -calculus" (invented around 1940).

Syntax of  $\lambda \rightarrow$ -Church

(type)  $A ::= p \mid A_1 \rightarrow A_2$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$

# Simply typed lambda calculus

Type environment:  $\Gamma ::= \cdot \mid \Gamma; x:A$

Typing rules:  $\Gamma \vdash e : A$  meaning under environment  $\Gamma$ ,  $e$  is well-typed and has type  $A$ .

$$x : A \vdash x : A \quad (\text{base})$$
$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow^i$$
$$\frac{\Gamma \vdash e_2 : A \quad \Delta \vdash e_1 : A \rightarrow B}{\Gamma, \Delta \vdash e_1 e_2 : B} \rightarrow^e$$

# Simply typed lambda calculus

Another variation:

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

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i$$

$$\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

Erase the terms, we get the “implicational fragment” of the propositional intuitionistic logic

# Some important properties

**Subject reduction** ( $\rightarrow_{\beta}$  preserves typing) :

if  $e \rightarrow_{\beta} e'$  and  $\Gamma \vdash e:A$  then  $\Gamma \vdash e':A$  .

**Unique typing**: if  $\Gamma \vdash e:A$ ,  $\Gamma \vdash e:A'$  then  $A \equiv A'$ .

**Strong-normalization**: if  $\Gamma \vdash e:A$ , then  $e$  is strongly normalizing.

**Limitation**: some perfectly fine terms  
 $(\lambda x. x)(\lambda x. x)$  is no longer supported.

**Challenge**: how to make the type system less constraining so that we can accept more terms.

# The Curry-Howard correspondence

Curry 1956: types-of-combinators / axioms in Hilbert-presentations

Howard 1969: lambda calculus / natural deduction  
(published in 1980)

One-to-one correspondence between typed  $\lambda$ -calculus and propositional logic:

- terms vs. proofs
- types vs. propositions
- function type  $\rightarrow$  vs. logic operator  $\rightarrow$
- typing rules vs. inference rules (natural deduction in sequent style)

The correspondence extends to all operators:  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\perp$ ,  
..... (and classic logic)

# Propositional intuitionistic logic

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \wedge_i \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{e1} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{e2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{i1}$$

$$\frac{\Gamma \vdash A \vee B \quad \Delta, A \vdash C \quad \Theta, B \vdash C}{\Gamma, \Delta, \Theta \vdash C} \vee_e$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{i2}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_i$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B} \rightarrow_e$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg_i$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash \neg A}{\Gamma, \Delta \vdash \perp} \neg_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_e$$

# A variation

basic sequent:  $\Gamma \vdash A$  if  $A \in \Gamma$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge i$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge e_1$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge e_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee i_1$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee e$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee i_2$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow i$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \rightarrow B}{\Gamma \vdash B} \rightarrow e$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg i$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \perp} \neg e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp e$$

# Extended typed lambda calculus

Syntax of  $\lambda \rightarrow$ -Church-Extended:

(type)  $A ::= p \mid \perp \mid A_1 \rightarrow A_2 \mid \neg A$   
 $\mid A_1 \wedge A_2 \mid A_1 \vee A_2$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \lambda^c x:A.e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e)$   
 $\mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e)$   
 $\mid \text{inj}_1(e) \mid \text{inj}_2(e)$   
 $\mid \text{case}(e, x_1.e_1, x_2.e_2)$

# Extended typed lambda calculus

With more familiar notations:

(type)  $A ::= p \mid \text{void} \mid A_1 \rightarrow A_2 \mid A \text{ cont}$   
 $\mid A_1^* A_2 \mid A_1 + A_2$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \lambda^c x:A.e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e)$   
 $\mid (e_1, e_2) \mid \text{fst}(e) \mid \text{snd}(e)$   
 $\mid \text{inj}_1(e) \mid \text{inj}_2(e)$   
 $\mid \text{case}(e, x_1.e_1, x_2.e_2)$

# Selected typing rules

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A$

basic sequent:  $\Gamma \vdash x : A$  if  $(x, A) \in \Gamma$

$$\frac{\Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : B}{\Gamma \vdash (e_1, e_2) : A \wedge B} \wedge_i \quad \frac{\Gamma \vdash e : A \wedge B}{\Gamma \vdash \pi_1(e) : A} \wedge_{e_1} \quad \frac{\Gamma \vdash e : A \wedge B}{\Gamma \vdash \pi_2(e) : B} \wedge_{e_2}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \text{inj}_1(e) : A \vee B} \vee_{i_1} \quad \frac{\Gamma \vdash e : B}{\Gamma \vdash \text{inj}_2(e) : A \vee B} \vee_{i_2}$$

$$\frac{\Gamma \vdash e : A \vee B \quad \Gamma, x_1 : A \vdash e_1 : C \quad \Gamma, x_2 : B \vdash e_2 : C}{\Gamma \vdash \text{case}(e, x_1.e_1, x_2.e_2) : C} \vee_e$$

# Selected typing rules

$$\frac{\Gamma, x:A \vdash B}{\Gamma \vdash \lambda x:A. e : A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Gamma, x:A \vdash e : \perp}{\Gamma \vdash \lambda^c x:A. e : \neg A} \neg_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : \neg A}{\Gamma \vdash \text{throw}(e_1, e_2) : \perp} \neg_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash e : \perp}{\Gamma \vdash \text{cast}(e) : A} \perp_e$$

# Part III: Predicate Logic

- Predicate logic
  - Definition of predicate languages
  - Predicate symbols; terms; variables; quantifiers
  - Natural deduction rules (also rules in sequent style)
  - Semantics of predicate logic
- Curry-Howard isomorphism
  - Typed lambda calculus revisited

# Propositional language revisited

A *propositional language*  $L = (P, O)$  :

- $P$  is a set of atomic sentences:  
 $\{p, q, r, \dots \dots \}$
- $O$  is a set of operators for putting them together:

and	$\wedge$	(arity 2)
or	$\vee$	(arity 2)
implies	$\rightarrow$	(arity 2)
not	$\neg$	(arity 1)
falsity	$\perp$	(arity 0)

# Predicate languages

Problem with propositional logic: it really didn't talk about any real-life entities.

What we need to add:

- domains: int, real, ...
- constants: 1, 3, 10, ...
- variables:  $x, y, \dots$
- functions:  $+, -, *, \div, \dots$
- predicates:  $\leq, \dots$
- formulas:  $x \leq y + 1, \dots$
- quantifiers:  $\forall, \exists$

# Predicate vocabulary

A predicate vocabulary consists of three sets:

- A set of **predicate symbols**  $\mathcal{P}$ .
- A set of **function symbols**  $\mathcal{F}$ .
- A set of **constant symbols**  $\mathcal{C}$ .

Each symbol comes with an arity.

**Terms** consist of (1) variables, (2) constants from  $\mathcal{C}$ , and (3) if  $f(t_1, t_2, \dots, t_n)$  where  $f \in \mathcal{F}$ , and  $t_1, t_2, \dots, t_n$  are also terms

# Predicate languages: terms

Write down its grammar:

(terms)  $t ::= x \mid c \mid f(t_1, \dots, t_n)$

where  $x$  is a variable,  $c \in \mathcal{C}$ , &  $f \in \mathcal{F}$  has arity  $n$ .

We often merge  $\mathcal{C}$  and  $\mathcal{F}$  by treating all constants as 0-arity functions.

So a predicate vocabulary is just a pair  $(\mathcal{F}, \mathcal{P})$

# Predicate languages: formulas

The set of formulas over  $(\mathcal{F}, \mathcal{P})$  consists of the following:

- $P(t_1, t_2, \dots, t_n)$  where  $P \in \mathcal{P}$ , and  $t_1, \dots, t_n$  are terms over  $\mathcal{F}$ , and the arity of  $P$  is  $n$  ( $n \geq 1$ ).
- $\perp$  is a formula.
- If  $A$  is a formula, then so is  $\neg A$ .
- If  $A_1$  and  $A_2$  are formulas, then so are  $(A_1 \wedge A_2)$ ,  $(A_1 \vee A_2)$ ,  $(A_1 \rightarrow A_2)$ .
- If  $A$  is a formula,  $x$  is a variable, then so are  $(\forall x.A)$  and  $(\exists x.A)$ .

# Predicate languages

A summary:

(terms)  $t ::= x \mid f(t_1, \dots, t_n)$  where  $f \in \mathcal{F}$   
(formulas)  $A ::= P(t_1, \dots, t_n) \mid \perp \mid \neg A$   
 $\mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \rightarrow A_2$   
 $\mid \forall x.A \mid \exists x.A$  where  $P \in \mathcal{P}$

- Notice in  $\forall x.A$  and  $\exists x.A$ , we didn't specify a "domain" for  $x$  --- this is not very general !!!

# Natural deduction

The natural-deduction rules for predicate logic is same as those for propositional logic.

At each stage of a proof, can introduce any formula as hypothesis (which can be discharged later by other rules).

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ B \end{array}}}{A \rightarrow B} \rightarrow i \qquad \frac{A \quad A \rightarrow B}{B} \rightarrow e$$

# Natural deduction

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

$$\frac{A \wedge B}{A} \wedge e_1 \quad \frac{A \wedge B}{B} \wedge e_2$$

$$\frac{A}{A \vee B} \vee i_1 \quad \frac{B}{A \vee B} \vee i_2$$

$$\frac{A \vee B \quad \boxed{\begin{array}{c} A \\ \vdots \\ C \end{array}} \quad \boxed{\begin{array}{c} B \\ \vdots \\ C \end{array}}}{C} \vee e$$

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ \perp \end{array}}}{\neg A} \neg i$$

$$\frac{A \quad \neg A}{\perp} \neg e$$

no intro rule for  $\perp$

$$\frac{\perp}{A} \perp e$$

# Natural deduction

New rules for the two quantifiers:

$$\frac{\boxed{\begin{array}{c} y \\ \vdots \\ A(y) \end{array}}}{\forall x.A(x)} \forall i$$

$$\frac{\forall x.A(x)}{A(t)} \forall e$$

$$\frac{A(t)}{\exists x.A(x)} \exists i$$

$$\frac{\exists x.A(x) \quad \boxed{\begin{array}{c} y \quad A(y) \\ \vdots \\ B \end{array}}}{B} \exists e$$

Note: In  $\forall i$  and  $\exists e$ , variable  $y$  doesn't occur free outside the box

# Natural deduction in sequent style

basic sequent:  $\Gamma \vdash A$  if  $A \in \Gamma$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge i$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge e_1$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge e_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee i_1$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee e$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee i_2$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow i$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \rightarrow B}{\Gamma \vdash B} \rightarrow e$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg i$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \perp} \neg e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp e$$

# Natural deduction in sequent style

$$\frac{x \in \Gamma}{\Gamma \vdash x} \quad \frac{\Gamma \vdash t_i \quad (1 \leq i \leq n) \quad f \in \mathcal{F} \quad f \text{ has arity } n}{\Gamma \vdash f(t_1, \dots, t_n)}$$

$$\frac{\Gamma, y \vdash A(y)}{\Gamma \vdash \forall x. A(x)} \forall i$$

$$\frac{\Gamma \vdash \forall x. A(x) \quad \Gamma \vdash t}{\Gamma \vdash A(t)} \forall e$$

$$\frac{\Gamma \vdash t \quad \Gamma \vdash A(t)}{\Gamma \vdash \exists x. A(x)} \exists i$$

$$\frac{\Gamma \vdash \exists x. A(x) \quad \Gamma, y, A(y) \vdash B}{\Gamma \vdash B} \exists e$$

# Extended typed $\lambda$ -calculus revisited

Let's review  $\lambda \rightarrow$ -Church-Extended:

(type)  $A ::= p \mid \perp \mid A_1 \rightarrow A_2 \mid \neg A$   
 $\mid A_1 \wedge A_2 \mid A_1 \vee A_2$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \lambda^c x:A.e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e)$   
 $\mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e)$   
 $\mid \text{inj}_1(e) \mid \text{inj}_2(e)$   
 $\mid \text{case}(e, x_1.e_1, x_2.e_2)$

# Extended typed $\lambda$ -calculus modified

Let's construct  $\lambda$ Pred-Church-Extended:

(type)  $A ::= P(t_1, \dots, t_n) \mid \perp \mid A_1 \rightarrow A_2$   
 $\mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \forall x. A \mid \exists x. A$

(exp)  $e ::= z \mid \lambda z:A. e \mid e_1 e_2$   
 $\mid \lambda^c z:A. e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e)$   
 $\mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e)$   
 $\mid \text{inj}_1(e) \mid \text{inj}_2(e) \mid \text{case}(e, z_1.e_1, z_2.e_2)$   
 $\mid \lambda x. e \mid e[t] \mid \text{pack}(t, e:A(t))$   
 $\mid \text{open } e_1 \text{ as } (x, z) \text{ in } e_2$

(term)  $t ::= x \mid f(t_1, \dots, t_n)$

# Selected typing rules

type environment:  $\Gamma ::= \cdot \mid \Gamma, z : A \mid \Gamma, x$

basic sequent:  $\Gamma \vdash z : A$  if  $(z, A) \in \text{Dom}(\Gamma)$

$$\frac{\Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : B}{\Gamma \vdash (e_1, e_2) : A \wedge B} \wedge_i \quad \frac{\Gamma \vdash e : A \wedge B}{\Gamma \vdash \pi_1(e) : A} \wedge_{e_1} \quad \frac{\Gamma \vdash e : A \wedge B}{\Gamma \vdash \pi_2(e) : B} \wedge_{e_2}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \text{inj}_1(e) : A \vee B} \vee_{i_1} \quad \frac{\Gamma \vdash e : B}{\Gamma \vdash \text{inj}_2(e) : A \vee B} \vee_{i_2}$$

$$\frac{\Gamma \vdash e_1 : A \vee B \quad \Gamma, z_1 : A \vdash e_1 : C \quad \Gamma, z_2 : B \vdash e_2 : C}{\Gamma \vdash \text{case}(e, z_1.e_1, z_2.e_2) : C} \vee_e$$

# Selected typing rules

$$\frac{\Gamma, z:A \vdash B}{\Gamma \vdash \lambda z:A. e : A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Gamma, z:A \vdash e : \perp}{\Gamma \vdash \lambda^c z:A. e : \neg A} \neg_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : \neg A}{\Gamma \vdash \text{throw}(e_1, e_2) : \perp} \neg_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash e : \perp}{\Gamma \vdash \text{cast}(e) : A} \perp_e$$

# Selected typing rules

Here are the typing rules for terms,  $\forall$  and  $\exists$

$$\frac{x \in \Gamma}{\Gamma \vdash x} \quad \frac{\Gamma \vdash t_i \quad (1 \leq i \leq n) \quad f \in \mathcal{F} \quad f \text{ has arity } n}{\Gamma \vdash f(t_1, \dots, t_n)}$$

$$\frac{\Gamma, x \vdash e : A(x)}{\Gamma \vdash \lambda x. e : \forall x. A(x)} \forall i \quad \frac{\Gamma \vdash e : \forall x. A(x) \quad \Gamma \vdash t}{\Gamma \vdash e[t] : A(t)} \forall e$$

$$\frac{\Gamma \vdash t \quad \Gamma \vdash e : A(t)}{\Gamma \vdash \text{pack}(t, e : A(t)) : \exists x. A(x)} \exists i$$

$$\frac{\Gamma \vdash e_1 : \exists x. A(x) \quad \Gamma, y, z : A(y) \vdash e_2 : B}{\Gamma \vdash \text{open } e_1 \text{ as } (y, z) \text{ in } e_2 : B} \exists e$$

# Definition of predicate languages

A predicate lang.  $L$  is a tuple  $(\mathcal{P}, \mathcal{T}, \mathcal{V}, \mathcal{O}, \mathcal{Q})$  where

- $\mathcal{P}$  is a set of predicate symbols, each associated with a non-negative integer (its arity).
- $\mathcal{T}$  is a set of terms (constructed using variables from  $\mathcal{V}$  and function symbols from  $\mathcal{F}$ ).
- $\mathcal{V}$  is a set of variables.
- $\mathcal{O}$  is a set of operators (e.g.,  $\perp, \neg, \wedge, \vee, \rightarrow$ ), each with a specified arity.
- $\mathcal{Q}$  is a set of quantifiers (e.g.,  $\forall, \exists$ )

# Semantics of predicate logic

A valuation system for a predicate language  $L$  is a tuple  $(M, D, F, G)$  where

- $M$  is a set with at least two elements and  $D$  is a non-empty proper subset of  $M$ .
- $F = (f_{o_1}, f_{o_2}, \dots, f_{o_n})$  is a set of functions, one for each operator in  $O = \{o_1, o_2, \dots, o_n\}$ , such that  $f_o : M^n \rightarrow M$  (where  $n$  is the arity of  $o$ ). We say  $f_o$  interprets  $o$ .
- $G$  is a set of functions from  $2^M$  to  $M$ , one corresponding to each quantifier in  $Q$ . The function  $g_q$  corresponds to the quantifier  $q$

# Semantics of predicate logic

Predicate language for classical first-order logic:  $\mathcal{O} = \{\wedge, \vee, \rightarrow, \neg, \perp\}$  and  $\mathcal{Q} = \{\forall, \exists\}$ .

The following valuation system for this gives classical logic:

- $M = \{\mathbf{t}, \mathbf{f}\}$  and  $D = \{\mathbf{t}\}$
- $F$  is defined as in the propositional case
- $G = \{g_{\forall}, g_{\exists}\}$  where

$$\begin{aligned} g_{\forall}(X) &= \mathbf{f} \quad \text{if } f \in X \\ g_{\forall}(X) &= \mathbf{t} \quad \text{otherwise} \end{aligned}$$

$$\begin{aligned} g_{\exists}(X) &= \mathbf{t} \quad \text{if } t \in X \\ g_{\exists}(X) &= \mathbf{f} \quad \text{otherwise} \end{aligned}$$

# Semantics of predicate logic

An assignment under a valuation system  $(M, D, F, G)$  for a language  $L = (\mathcal{P}, \mathcal{T}, \mathcal{V}, \mathcal{O}, \mathcal{Q})$  is a pair  $(a, I)$  where  $a$  is a function and  $I$  is a non-empty set such that:

- If  $t \in \mathcal{T}$  then  $a(t)$  is an element of  $I$ .
- If  $x \in \mathcal{V}$  then  $a(x) \in I$ .
- If  $P \in \mathcal{P}$  has arity  $n$  then  $a(P)$  is a function from the set  $I^n$  to the set  $M$  (i.e.,  $a(P) : I^n \rightarrow M$ )

Essentially, for each  $f \in \mathcal{F}$  with arity  $n$ ,  $a(f)$  is a function from the set  $I^n$  to the set  $I$   
(i.e.,  $a(f) : I^n \rightarrow I$ )

# Semantics of predicate logic

Each assignment  $(a, I)$  induces an interpretation  $v_a$ :

- If  $p \in P$  has arity  $n$  and  $t_1, \dots, t_n \in T$  then  $v_a(p(t_1, \dots, t_n)) = a(p)(a(t_1), \dots, a(t_n))$ .
- $v_a(o(A_1, \dots, A_n)) = f_o(v_a(A_1), \dots, v_a(A_n))$  where  $n$  is the arity of  $o$ .
- $v_a(q \times A) = g_q(\{v_{a'}(A) \mid a' = a[x_i \mapsto i] \text{ for all } i \in I\})$

An interpretation is a valuation system plus an assignment.

# Models [Huth & Ryan]

Let  $\mathcal{F}$  be a set of function symbols and  $\mathcal{P}$  a set of predicate symbols, each symbol with a fixed number of required arguments. A model  $\mathcal{M}$  of the pair  $(\mathcal{F}, \mathcal{P})$  consists of the following set of data:

- A non-empty set  $I$ , the universe of concrete values
- For each  $f \in \mathcal{F}$  with  $n$  arguments a concrete function  $f^{\mathcal{M}} : I^n \rightarrow I$  from  $I^n$ , the set of  $n$ -tuples over  $A$ , to  $A$ ; and
- For each  $P \in \mathcal{P}$  with  $n$  arguments a subset  $P^{\mathcal{M}} \subseteq I^n$  of  $n$ -tuples over  $A$ .

# Environment [Huth & Ryan]

An environment  $I$  is a mapping from every variable  $x$  (in  $\mathcal{V}$ ) to a value  $I(x)$  of  $\mathcal{I}$ .

Given a model  $\mathcal{M}$  for a pair  $(\mathcal{F}, \mathcal{P})$  and given an environment  $I$ , we can build the assignment  $(a, I)$  and a corresponding interpretation  $v_a$ .

From  $v_a$ , we can define a semantic entailment relation  $\Gamma \models^{v_a} A$ .

We write  $\Gamma \models A$  if  $\Gamma \models^{v_a} A$  is true for all  $(a, I)$  (or for all  $(\mathcal{M}, I)$  )

# Important properties

A presentation  $\vdash$  is **sound with respect to a semantics**  $\models$  if for all  $\Gamma$  and  $A$ , if  $\Gamma \vdash A$ , then  $\Gamma \models A$ .

A presentation  $\vdash$  is **complete with respect to a semantics**  $\models$  if for all  $\Gamma$  and  $A$ , if  $\Gamma \models A$ , then  $\Gamma \vdash A$ .

Given a presentation  $\vdash$ , a set of formulas  $\Gamma$  is inconsistent iff for some formula  $A$ , we have both  $\Gamma \vdash A$  and  $\Gamma \vdash \neg A$ .

A presentation  $\vdash$  is **negation consistent** if there is no  $A$  such that  $\emptyset \vdash A$  and  $\emptyset \vdash \neg A$ .

# Important properties

A presentation  $\vdash$  is **absolute consistency** if there exists an  $A$  such that  $\emptyset \not\vdash A$ .

For classic/intuitive logics, **negation consistency** and **absolute consistency** are equivalent.

Given a presentation  $\vdash$ , a set of formulas  $\Gamma$  is **complete** if for every closed formula  $A$ , either  $\Gamma \vdash A$  or  $\Gamma \vdash \neg A$ .

[This is not same as "**complete w. r. t. a semantics  $\models$** ".]

Inconsistent logics are always complete! But they are useless!

# Part IV: Logic vs. Typed Lambda Calculi

- Propositional logic (1<sup>st</sup> order, 2<sup>nd</sup> order, ... higher-order)
- Typed  $\lambda$ -calculi ( $\lambda \rightarrow$ , System F,  $F_\omega$ )
- Predicate logic (1<sup>st</sup> order, 2<sup>nd</sup> order, ... higher-order)
- Many-sorted predicate logic
- Dependently typed  $\lambda$ -calculi

# Propositional languages

A *propositional language*  $L = (P, O)$  :

- $P$  is a set of atomic sentences:  
 $\{p, q, r, \dots \dots \}$
- $O$  is a set of operators for putting them together:

and	$\wedge$	(arity 2)
or	$\vee$	(arity 2)
implies	$\rightarrow$	(arity 2)
not	$\neg$	(arity 1)
falsity	$\perp$	(arity 0)

# First-order propositional logic

Natural deduction rules:

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ B \end{array}}}{A \rightarrow B} \rightarrow i$$

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

no intro rule for  $\perp$

$$\frac{\perp}{A} \perp e$$

# First-order propositional logic

Natural deduction rules in sequent style:

basic sequent:  $\Gamma \vdash A$  if  $A \in \Gamma$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash A \quad \Gamma \vdash A \rightarrow B}{\Gamma \vdash B} \rightarrow_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_e$$

# Simply typed lambda calculus

Syntax for the simply typed  $\lambda$  calculus:

(type)  $A ::= p \mid \perp \mid A_1 \rightarrow A_2$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \text{cast}(e)$

# Typing rules for simply typed $\lambda$ -calculus

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A$

basic sequent:  $\Gamma \vdash x : A$  if  $(x, A) \in \Gamma$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

no intro for  $\perp$

$$\frac{\Gamma \vdash e : \perp}{\Gamma \vdash \text{cast}(e) : A} \perp_e$$

# Second order propositional language

We add universal quantification over atomic sentences:

$$\text{(props)} \quad A ::= p \mid \perp \mid A_1 \rightarrow A_2 \\ \mid \forall p.A$$

$$\frac{\boxed{\begin{array}{c} p \\ \vdots \\ A(p) \end{array}}}{\forall p.A(p)} \forall i \quad \frac{\forall p.A(p) \quad A'}{A(A')} \forall e$$

# Polymorphically typed lambda calculus

Syntax of system F (a.k.a., 2<sup>nd</sup> order polymorphically typed  $\lambda$  calculus):

(type)  $A ::= p \mid \perp \mid A_1 \rightarrow A_2 \mid \forall p. A$

(term)  $e ::= x \mid \lambda x:A. e \mid e_1 e_2$   
|  $\text{cast}(e)$   
|  $\lambda p. e \mid e(A)$

# Polymorphically typed lambda calculus

No need for  $\perp$  as it can be represented as  $\forall p.p$

Syntax of system F:

(type)  $A ::= p \mid A_1 \rightarrow A_2 \mid \forall p.A$

(term)  $e ::= x \mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \lambda p.e \mid e(A)$

# Typing rules for system F

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, p$

basic sequents:  $\Gamma \vdash x : \Gamma(x)$        $\Gamma \vdash p$  if  $p \in \Gamma$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \rightarrow B}$$

$$\frac{\Gamma, p \vdash A}{\Gamma \vdash \forall p. A}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Gamma, p \vdash e : A}{\Gamma \vdash \lambda p. e : \forall p. A} \forall_i$$

$$\frac{\Gamma \vdash e : \forall p. A \quad \Gamma \vdash A'}{\Gamma \vdash e [A'] : [A'/p]A} \forall_e$$

# System F with kind annotations

We can make up a “kind” for all types

(kind)  $K ::= \Omega$

(type)  $A ::= p \mid A_1 \rightarrow A_2 \mid \forall p:K. A$

(term)  $e ::= x \mid \lambda x:A. e \mid e_1 e_2$   
 $\mid \lambda p:K. e \mid e(A)$

# System F with kind annotations

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, p : K$

basic sequents:  $\Gamma \vdash x : \Gamma(x)$        $\Gamma \vdash p : \Gamma(p)$  if  $p \in \Gamma$

$$\frac{\Gamma \vdash A : \Omega \quad \Gamma \vdash B : \Omega}{\Gamma \vdash A \rightarrow B : \Omega}$$

$$\frac{\Gamma, p : K \vdash A : \Omega}{\Gamma \vdash \forall p : K. A : \Omega}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i$$

$$\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Gamma, p : K \vdash e : A}{\Gamma \vdash \lambda p : K. e : \forall p : K. A} \forall_i$$

$$\frac{\Gamma \vdash e : \forall p : K. A \quad \Gamma \vdash A' : K}{\Gamma \vdash e [A'] : [A'/p]A} \forall_e$$

# A variant: splitting the environment

kind environment:  $\Delta ::= \cdot \mid \Delta, p : K$

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A$

basic sequents:  $\Delta \vdash p : \Delta(p)$      $\Delta; \Gamma \vdash x : \Gamma(x)$

$$\frac{\Delta \vdash A : \Omega \quad \Delta \vdash B : \Omega}{\Delta \vdash A \rightarrow B : \Omega}$$

$$\frac{\Delta, p : K \vdash A : \Omega}{\Delta \vdash \forall p : K. A : \Omega}$$

$$\frac{\Delta; \Gamma, x : A \vdash B}{\Delta; \Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i \quad \frac{\Delta; \Gamma \vdash e_2 : A \quad \Delta; \Gamma \vdash e_1 : A \rightarrow B}{\Delta; \Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Delta, p : K; \Gamma \vdash e : A}{\Delta; \Gamma \vdash \lambda p : K. e : \forall p : K. A} \forall_i \quad \frac{\Delta; \Gamma \vdash e : \forall p : K. A \quad \Delta \vdash A' : K}{\Delta; \Gamma \vdash e [A'] : [A'/p]A} \forall_e$$

# Higher-order propositional language

We can also add higher-order predicates:

(prop-kind)  $K ::= \text{Prop} \mid K_1 \rightarrow K_2$


(props)  $A ::= p \mid A_1 \rightarrow A_2$   
 $\mid \forall p:K. A \mid \lambda p:K. A \mid A_1(A_2)$

$$\frac{\boxed{\begin{array}{c} p : K \\ \vdots \\ A(p) \end{array}}}{\forall p : K. A(p)} \forall i \quad \frac{\forall p : K. A(p) \quad A' : K}{A(A')} \forall e$$

# Higher-order polymorphic $\lambda$ -calculus

Add type-functions to **system F**, we get **F<sub>ω</sub>**

(kind)  $K ::= \Omega \mid K_1 \rightarrow K_2$



(type)  $A ::= p \mid A_1 \rightarrow A_2 \mid \forall p:K. A$   
 $\mid \lambda p:K. A \mid A_1(A_2)$

(term)  $e ::= x \mid \lambda x:A. e \mid e_1 e_2$   
 $\mid \lambda p:K. e \mid e(A)$

# Typing rules for $F_{\omega}$

type environment:  $\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, p : K$

basic sequents:  $\Gamma \vdash x : \Gamma(x)$        $\Gamma \vdash p : \Gamma(p)$  if  $p \in \Gamma$

$$\frac{\Gamma \vdash A : \Omega \quad \Gamma \vdash B : \Omega}{\Gamma \vdash A \rightarrow B : \Omega}$$

$$\frac{\Gamma, p : K \vdash A : \Omega}{\Gamma \vdash \forall p : K. A : \Omega}$$

$$\frac{\Gamma, p : K_1 \vdash A : K_2}{\Gamma \vdash \lambda p : K_1. A : K_1 \rightarrow K_2}$$

$$\frac{\Gamma \vdash A : K_1 \rightarrow K_2 \quad \Gamma \vdash B : K_1}{\Gamma \vdash A(B) : K_2}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow_i$$

$$\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \rightarrow_e$$

$$\frac{\Gamma, p : K \vdash e : A}{\Gamma \vdash \lambda p : K. e : \forall p : K. A} \forall_i$$

$$\frac{\Gamma \vdash e : \forall p : K. A \quad \Gamma \vdash A' : K}{\Gamma \vdash e [A'] : [A'/p]A} \forall_e$$

# First-order predicate languages

A simplified version:

(terms)  $t ::= x \mid f(t_1, \dots, t_n)$  where  $f \in \mathcal{F}$

(formulas)  $A ::= P(t_1, \dots, t_n) \mid \perp \mid A_1 \rightarrow A_2$   
 $\mid \forall x.A$  where  $P \in \mathcal{P}$

# Natural deduction

New rules for the two quantifiers:

$$\frac{\boxed{\begin{array}{c} y \\ \vdots \\ A(y) \end{array}}}{\forall x.A(x)} \forall i$$

$$\frac{\forall x.A(x)}{A(t)} \forall e$$

$$\frac{A(t)}{\exists x.A(x)} \exists i$$

$$\frac{\exists x.A(x) \quad \boxed{\begin{array}{c} y \quad A(y) \\ \vdots \\ B \end{array}}}{B} \exists e$$

Note: In  $\forall i$  and  $\exists e$ , variable  $y$  doesn't occur free outside the box

# Natural deduction in sequent style

$$\frac{x \in \Gamma}{\Gamma \vdash x} \quad \frac{\Gamma \vdash t_i \quad (1 \leq i \leq n) \quad f \in \mathcal{F} \quad f \text{ has arity } n}{\Gamma \vdash f(t_1, \dots, t_n)}$$

$$\frac{\Gamma, y \vdash A(y)}{\Gamma \vdash \forall x. A(x)} \forall i$$

$$\frac{\Gamma \vdash \forall x. A(x) \quad \Gamma \vdash t}{\Gamma \vdash A(t)} \forall e$$

$$\frac{\Gamma \vdash t \quad \Gamma \vdash A(t)}{\Gamma \vdash \exists x. A(x)} \exists i$$

$$\frac{\Gamma \vdash \exists x. A(x) \quad \Gamma, y, A(y) \vdash B}{\Gamma \vdash B} \exists e$$

# Minimized $\lambda$ Pred

A minimized version of  $\lambda$ Pred:

(type)  $A ::= P(t_1, \dots, t_n) \mid \perp$   
 $\mid A_1 \rightarrow A_2 \mid \forall x. A$

(exp)  $e ::= z \mid \lambda z:A. e \mid e_1 e_2$   
 $\mid \text{cast}(e) \mid \lambda x. e \mid e[t]$

(term)  $t ::= x \mid f(t_1, \dots, t_n)$

# Higher-order predicate languages

(tm-types)  $T ::= \text{Int} \mid \text{Bool} \mid T_1 \rightarrow T_2$

(terms)  $t ::= x \mid f(t_1, \dots, t_n) \quad \text{where } f \in \mathcal{F}$   
 $\mid \lambda x:T.t \mid t_1(t_2)$

(prop-kind)  $K ::= \text{Prop} \mid K_1 \rightarrow K_2$

(formulas)  $A ::= P(t_1, \dots, t_n) \mid A_1 \rightarrow A_2$   
 $\mid \forall x:T.A \quad \text{where } P \in \mathcal{P}$   
 $\mid p \mid \forall p:K.A \mid \lambda p:K.A \mid A_1(A_2)$

(prf-terms)  $e ::= z \mid \lambda z:A.e \mid e_1 e_2 \mid \lambda x:T.e \mid e(t)$   
 $\mid \lambda p:K.e \mid e(A)$

# Predicate languages simplified

We'll add back function and predicate symbols via inductive definitions later:

(tm-kind)  $U ::= \text{Set}$

(tm-types)  $T ::= \alpha \mid T_1 \rightarrow T_2$

(terms)  $t ::= x \mid \lambda x:T.t \mid t_1(t_2)$

(prop-kind)  $K ::= \text{Prop} \mid K_1 \rightarrow K_2 \mid \Pi x:T.K$

(formulas)  $A ::= p \mid A_1 \rightarrow A_2 \mid \lambda x:T.A \mid A(t)$   
 $\mid \forall x:T.A \mid p \mid \forall p:K.A \mid \lambda p:K.A \mid A_1(A_2)$

(prf-terms)  $e ::= z \mid \lambda z:A.e \mid e_1 e_2 \mid \lambda x:T.e \mid e(t)$   
 $\mid \lambda p:K.e \mid e(A)$

# Predicate lang w. polymorphic terms

(tm-kind)  $U ::= \text{Set} \quad | \quad U_1 \rightarrow U_2$

(tm-types)  $T ::= \alpha \quad | \quad T_1 \rightarrow T_2 \quad | \quad \forall \alpha:U. T \quad | \quad \lambda \alpha:U. T \quad | \quad T_1(T_2)$

(terms)  $t ::= x \quad | \quad \lambda x:T. t \quad | \quad t_1(t_2) \quad | \quad \lambda \alpha:U. t \quad | \quad t(T)$

(prop-kind)  $K ::= \text{Prop} \quad | \quad K_1 \rightarrow K_2 \quad | \quad \Pi x:T. K$

(formulas)  $A ::= p \quad | \quad A_1 \rightarrow A_2 \quad | \quad \lambda x:T. A \quad | \quad A(t)$   
 $\quad | \quad \forall x:T. A \quad | \quad p \quad | \quad \forall p:K. A \quad | \quad \lambda p:K. A \quad | \quad A_1(A_2)$

(prf-terms)  $e ::= z \quad | \quad \lambda z:A. e \quad | \quad e_1 e_2 \quad | \quad \lambda x:T. e \quad | \quad e(t)$   
 $\quad | \quad \lambda p:K. e \quad | \quad e(A)$

# Unifying $\rightarrow$ , $\forall$ , and $\Pi$

(tm-kind)  $U ::= \text{Set} \quad | \quad \Pi\alpha:U_1.U_2$

(tm-types)  $T ::= \alpha \quad | \quad \Pi x:T_1.T_2 \quad | \quad \Pi\alpha:U.T \quad | \quad \lambda\alpha:U.T \quad | \quad T_1(T_2)$

(terms)  $t ::= x \quad | \quad \lambda x:T.t \quad | \quad t_1(t_2) \quad | \quad \lambda\alpha:U.t \quad | \quad t(T)$

(prop-kind)  $K ::= \text{Prop} \quad | \quad \Pi p:K_1.K_2 \quad | \quad \Pi x:T.K$

(formulas)  $A ::= p \quad | \quad \Pi z:A_1.A_2 \quad | \quad \lambda x:T.A \quad | \quad A(t)$   
 $\quad | \quad \Pi x:T.A \quad | \quad p \quad | \quad \Pi p:K.A \quad | \quad \lambda p:K.A \quad | \quad A_1(A_2)$

(prf-terms)  $e ::= z \quad | \quad \lambda z:A.e \quad | \quad e_1 e_2 \quad | \quad \lambda x:T.e \quad | \quad e(t)$   
 $\quad | \quad \lambda p:K.e \quad | \quad e(A)$

# Seal the top universe

(tm-knd-scm)  $u ::= \text{Type}'$

(tm-kind)  $U ::= \text{Set} \quad | \quad \Pi\alpha:U_1.U_2$

(tm-types)  $T ::= \alpha \quad | \quad \Pi x:T_1.T_2 \quad | \quad \Pi\alpha:U.T \quad | \quad \lambda\alpha:U.T \quad | \quad T_1(T_2)$

(terms)  $t ::= x \quad | \quad \lambda x:T.t \quad | \quad t_1(t_2) \quad | \quad \lambda\alpha:U.t \quad | \quad t(T)$

(p-knd-scm)  $w ::= \text{Type}$

(prop-kind)  $K ::= \text{Prop} \quad | \quad \Pi p:K_1.K_2 \quad | \quad \Pi x:T.K$

(formulas)  $A ::= p \quad | \quad \Pi z:A_1.A_2 \quad | \quad \lambda x:T.A \quad | \quad A(t)$   
 $\quad | \quad \Pi x:T.A \quad | \quad p \quad | \quad \Pi p:K.A \quad | \quad \lambda p:K.A \quad | \quad A_1(A_2)$

(prf-terms)  $e ::= z \quad | \quad \lambda z:A.e \quad | \quad e_1 e_2 \quad | \quad \lambda x:T.e \quad | \quad e(t)$   
 $\quad | \quad \lambda p:K.e \quad | \quad e(A)$

# The big merge

(tm,p-kind-scm)  $u, w ::= \text{Type}' \mid \text{Type}$

(tm,p-kind)  $U, K ::= \text{Set} \mid \Pi\alpha:U_1.U_2$   
 $\mid \text{Prop} \mid \Pi p:K_1.K_2 \mid \Pi x:T.K$

(tm-types,form)  $T, A ::= \alpha \mid \Pi x:T_1.T_2 \mid \Pi\alpha:U.T \mid \lambda\alpha:U.T \mid T_1(T_2)$   
 $\mid p \mid \Pi z:A_1.A_2 \mid \lambda x:T.A \mid A(t)$   
 $\mid \Pi x:T.A \mid p \mid \Pi p:K.A \mid \lambda p:K.A \mid A_1(A_2)$

(terms,prf-tm)  $t, e ::= x \mid \lambda x:T.t \mid t_1(t_2) \mid \lambda\alpha:U.t \mid t(T)$   
 $\mid z \mid \lambda z:A.e \mid e_1 e_2 \mid \lambda x:T.e \mid e(t)$   
 $\mid \lambda p:K.e \mid e(A)$

# Calculus of constructions

(knd-scm)  $u ::= \text{Type}$

(kind)  $K ::= \text{Prop} \mid \Pi p:K_1.K_2 \mid \Pi x:A.K$

(type)  $A ::= p \mid \Pi x:A_1.A_2 \mid \Pi p:K.A$   
|  $\lambda p:K.A \mid A_1(A_2)$   
|  $\lambda x:A_1.A_2 \mid A(e)$

(term)  $e ::= x$   
|  $\lambda x:A.e \mid e_1 e_2$   
|  $\lambda p:K.e \mid e(A)$

With two  
sorts:



Four  $\Pi$ 's:

(Prop, Prop)

(Prop, Type)

(Type, Prop)

(Type, Type)

# Pure type systems

(ptm)  $T ::= s \mid x \mid \Pi x:T_1.T_2$   
 $\mid \lambda x:T_1.T_2 \mid T_1(T_2)$

- To rule out ill-formed terms, we define what are well-formed  $\Pi$ 's
- A set of sorts, e.g., Type, Prop
- A set of rules  $(s_1, s_2, s_3)$ : if  $T_1$  has sort  $s_1$ , and  $T_2$  has sort  $s_2$ , then  $\Pi x:T_1.T_2$  has sort  $s_3$
- Write  $(s_1, s_2)$  to mean  $(s_1, s_2, s_2)$

CC with two  
sorts:



Four  $\Pi$ 's:

$(\text{Prop}, \text{Prop})$

$(\text{Prop}, \text{Type})$

$(\text{Type}, \text{Prop})$

$(\text{Type}, \text{Type})$

# CC with four sorts

(ptm)

$$\begin{aligned} T ::= & s \mid x \mid \Pi x:T_1.T_2 \\ & \mid \lambda x:T_1.T_2 \mid T_1(T_2) \end{aligned}$$

**Sixteen  $\Pi$ 's:**

(Set, Set), (Type', Set), (Type', Type')

(Prop, Prop), (Type, Prop), (Type, Type),  
(Set, Type), (Set, Prop)

(Set, Type'), (Prop, Type), (Prop, Set), (Prop, Type')  
(Type', Type), (Type', Prop), (Type, Set), (Type, Type')

**CC with four  
sorts:**

Type    Type'  
|        |  
Prop    Set

# CC with three sorts

(ptm)

$$\begin{aligned} T ::= & s \mid x \mid \Pi x:T_1.T_2 \\ & \mid \lambda x:T_1.T_2 \mid T_1(T_2) \end{aligned}$$

**Nine  $\Pi$ 's:**

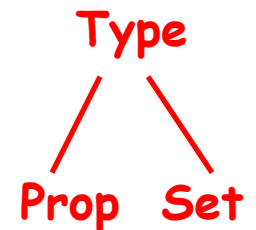
**(Set,Set), (Type,Set), (Type,Type),**

**(Prop,Prop), (Type,Prop)**

**(Set,Type), (Set,Prop)**

**(Prop,Set), (Prop,Type)**

**CC with three  
sorts:**



# CC with infinite number of universes

(ptm)

$$T ::= s \mid x \mid \Pi x:T_1.T_2 \\ \mid \lambda x:T_1.T_2 \mid T_1(T_2)$$

**Infinitely many  $\Pi$ 's:**

**(Set, Set), (TypeI, Set)**

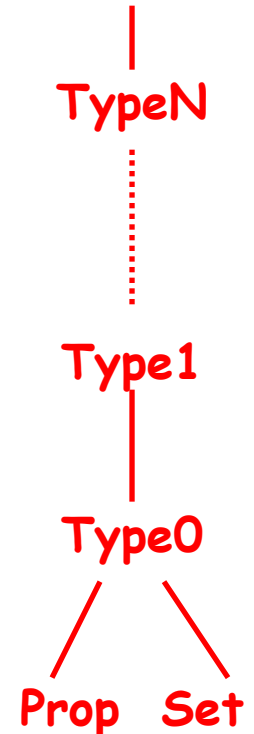
**(Prop, Prop), (TypeI, Prop)**

**(Set, TypeI), (Set, Prop)**

**(Prop, Set), (Prop, TypeI)**

**(TypeI, TypeJ, TypeK) where  $I \leq K$  and  $J \leq K$**

**CC with many sorts:**



# Adding inductive definitions

(tm-knd-scm)  $u ::= \text{Type}'$

(tm-kind)  $U ::= \text{Set} \mid \Pi\alpha:U_1.U_2$

(tm-types)  $T ::= \alpha \mid \Pi x:T_1.T_2 \mid \Pi\alpha:U.T \mid \lambda\alpha:U.T \mid T_1(T_2) \mid \text{Ind}(\alpha:U)(T_1, \dots, T_n)$

(terms)  $t ::= x \mid \lambda x:T.t \mid t_1(t_2) \mid \lambda\alpha:U.t \mid t(T) \mid \text{Ctor}(i, T)$

(p-knd-scm)  $w ::= \text{Type}$

(prop-kind)  $K ::= \text{Prop} \mid \Pi p:K_1.K_2 \mid \Pi x:T.K$

(formulas)  $A ::= p \mid \Pi z:A_1.A_2 \mid \lambda x:T.A \mid A(t) \mid \Pi x:T.A \mid p \mid \Pi p:K.A \mid \lambda p:K.A \mid A_1(A_2) \mid \text{Ind}(p:K)(A_1, \dots, A_n)$

(prf-terms)  $e ::= z \mid \lambda z:A.e \mid e_1 e_2 \mid \lambda x:T.e \mid e(t) \mid \lambda p:K.e \mid e(A) \mid \text{Ctor}(i, A)$

# CC with inductive definitions

(knd-scm)  $u ::= \text{Type}$

(kind)  $K ::= \text{Prop} \mid \Pi p:K_1.K_2 \mid \Pi x:A.K$

(type)  $A ::= p \mid \Pi x:A_1.A_2 \mid \Pi p:K.A$   
 $\mid \lambda p:K.A \mid A_1(A_2)$   
 $\mid \lambda x:A_1.A_2 \mid A(e) \mid \text{Ind}(p:K)(A_1, \dots, A_n)$

(term)  $e ::= x$   
 $\mid \lambda x:A.e \mid e_1 e_2$   
 $\mid \lambda p:K.e \mid e(A) \mid \text{Ctor}(i, A)$

With two  
sorts:



Four  $\Pi$ 's:

(Prop, Prop)

(Prop, Type)

(Type, Prop)

(Type, Type)

# Inductive definitions: syntactic sugar

$$\begin{array}{l} \text{Inductive } I : K = C_1 : A_1 \\ \quad | C_2 : A_2 \\ \quad \dots\dots\dots \\ \quad | C_n : A_n \end{array}$$

represents:

$$I \equiv \text{Ind}(p : K) ([p/I]A_1, \dots, [I/p]A_n)$$
$$C_1 \equiv \text{Ctor}(1, I)$$
$$C_2 \equiv \text{Ctor}(2, I)$$

.....

$$C_n \equiv \text{Ctor}(n, I)$$

**CiC also adds "CASE" and primitive recursion "FIX"**