Curry-Howard Isomorphism
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
A propositional language \( L \) contains

- a set of atomic sentences:
  \{p, q, r, \ldots \}

- a set of operators for putting them together:

\[
\begin{array}{ccc}
\text{and} & \land & \text{(arity 2)} \\
\text{or} & \lor & \text{(arity 2)} \\
\text{implies} & \rightarrow & \text{(arity 2)} \\
\text{not} & \neg & \text{(arity 1)} \\
\text{falsity} & \bot & \text{(arity 0)}
\end{array}
\]
Proof-based view of logic

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

\[ \Gamma \vdash A \]

where \( \Gamma = A_1, A_2, \ldots, A_n \)

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:

Ax1  \( A \rightarrow (B \rightarrow A) \)
Ax2  \( (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \)
Ax3  \( (\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A) \)

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

\[
\text{MP} \quad \begin{array}{c} A \rightarrow B \\ A \end{array} \quad B
\]
Hilbert-Frege presentations

The operators $\land$, $\lor$, and $\bot$ are derived forms:

$$ A \land B \equiv \neg(A \rightarrow \neg B) $$
$$ A \lor B \equiv \neg A \rightarrow B $$
$$ \bot \equiv \neg(A \rightarrow A) $$

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. $\neg q \rightarrow (\neg p \rightarrow \neg q)$ \hspace{1cm} Ax1
2. $(\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$ \hspace{1cm} Ax3
3. $((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)) \rightarrow$
   \hspace{1cm} $(\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)))$ \hspace{1cm} Ax1
4. $\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p))$ \hspace{1cm} MP 2,3
5. $((\neg q \rightarrow ((\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p))) \rightarrow$
   \hspace{1cm} $((\neg q \rightarrow (\neg p \rightarrow \neg q)) \rightarrow (\neg q \rightarrow (q \rightarrow p)))$ \hspace{1cm} Ax2
6. $(\neg q \rightarrow (\neg p \rightarrow \neg q)) \rightarrow (\neg q \rightarrow (q \rightarrow p))$ \hspace{1cm} MP 4,5
7. $\neg q \rightarrow (q \rightarrow p)$ \hspace{1cm} MP 1,6

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

\[
\begin{array}{c}
A \\
\vdots \\
B
\end{array} \\ \frac{A \rightarrow B}{\rightarrow i}
\]

\[
\frac{A}{\frac{A}{B}} \rightarrow e
\]
Natural deduction

\[ \frac{A}{A \land B} \land i \]
\[ \frac{\land e_1}{A} \]
\[ \frac{\land e_2}{B} \]
\[ \frac{A \lor B}{C} \lor e \]

\[ \frac{A}{\neg A} \neg i \]
\[ \frac{A}{\bot} \neg e \]

no intro rule for \( \bot \)
\[ \frac{\bot}{A} \bot e \]
Natural deduction

The previous set of rules define “propositional intuitionistic logic”

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

\[
\frac{\neg \neg A}{A} \quad \neg \neg
\]

This allows you to “prove by contradiction”
Natural deduction proof for $\emptyset \vdash \neg q \rightarrow (q \rightarrow p)$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$\neg q$</td>
<td>Assumption</td>
</tr>
<tr>
<td>2.</td>
<td>$q$</td>
<td>Assumption</td>
</tr>
<tr>
<td>3.</td>
<td>$\bot$</td>
<td>$\neg e$ 1,2</td>
</tr>
<tr>
<td>4.</td>
<td>$p$</td>
<td>$\bot e$ 3</td>
</tr>
<tr>
<td>5.</td>
<td>$q \rightarrow p$</td>
<td>$\rightarrow i$ 2-4</td>
</tr>
<tr>
<td>6.</td>
<td>$\neg q \rightarrow (q \rightarrow p)$</td>
<td>$\rightarrow i$ 1-5</td>
</tr>
</tbody>
</table>
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:

\[ \Gamma \vdash A \]
Natural deduction in sequent style

\[
\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A \land B} \quad \& i \\
\frac{\Delta \vdash B}{\Gamma, \Delta \vdash A \land B} \quad \& i
\]

\[
\frac{\Gamma \vdash A \land B}{\Gamma \vdash A} \quad \& e_1 \\
\frac{\Gamma \vdash A \land B}{\Gamma \vdash B} \quad \& e_2
\]

\[
\frac{\Gamma \vdash A}{\Gamma \vdash A \lor B} \quad \lor i_1 \\
\frac{\Gamma \vdash B}{\Gamma \vdash A \lor B} \quad \lor i_2
\]

\[
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \quad \rightarrow i
\]

\[
\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A \rightarrow B} \quad \rightarrow e
\]

\[
\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \quad \neg i
\]

\[
\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash \bot} \quad \neg e
\]

\[
\text{no intro for } \bot
\]

\[
\frac{\Gamma \vdash \bot}{\Gamma \vdash \bot} \quad \bot e
\]
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:
  \[
  \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} \quad \wedge i
  \]

- 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 \( \land, \lor, \rightarrow, \neg, \bot \).

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

\[ \Gamma \vDash A \]

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.
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
- $F = (f_{o_1}, f_{o_2}, ..., f_{o_n})$ is a set of functions, one for each operator in $O = \{o_1, o_2, ..., o_n\}$, such that $f_o : M^n \rightarrow M$ (where $n$ is the arity of $o$). We say $f_o$ interprets $o$. 
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 \( p \in P \)
- \( v_a(o(A_1,\ldots,A_n)) = f_o(v_a(A_1),\ldots,v_a(A_n)) \) 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:

\[ M = \{t, f\} \]

\[ D = \{t\} \]

\[ F = \{f_\wedge, f_\vee, f_\rightarrow, f_\neg, f_\bot\} \]

where

\[
\begin{array}{c|cc}
\wedge & t & f \\
\hline
 t & t & f \\
 f & f & f
\end{array}
\quad
\begin{array}{c|cc}
\vee & t & f \\
\hline
 t & t & t \\
 f & t & f
\end{array}
\quad
\begin{array}{c|cc}
\rightarrow & t & f \\
\hline
 t & t & f \\
 f & t & f
\end{array}
\quad
\begin{array}{c|c}
\neg & \\
\hline
 t & f \\
 f & t \\
 f & f
\end{array}
\quad
\begin{array}{c|c}
\bot & \\
\hline
 t & t \\
 f & t
\end{array}
\]
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 consistent** 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-Horward 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:

\[ ee_1...e_n = (((ee_1)e_2)...e_n) \]
\[ \lambda x_1...x_n. e = \lambda x_1.(\lambda x_2.(... (\lambda x_n.e)...)) \]

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

\[ \lambda x. e \equiv_\alpha \lambda y. [y/x]e \quad \text{if} \ y \not\in \text{FV}(e) \]
The set of free variables in $e$, denoted as $FV(e)$ is defined as:

- $FV(x) = \{x\}$;
- $FV(e_1e_2) = FV(e_1) \cup FV(e_2)$;
- $FV(\lambda x.e) = FV(e) - \{x\}$.

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

The substitution \([e'/x]e\) is defined as:

\[
egin{align*}
[e'/x] x &= e' \\
[e'/x] y &= y \\
[e'/x] (e_1 e_2) &= ([e'/x] e_1)([e'/x] e_2) \\
[e'/x] (\lambda x. e_1) &= \lambda x. e_1 \\
[e'/x] (\lambda y. e_1) &= \lambda y. [e'/x] e_1 \quad \text{if } x \notin FV(e_1). \\
[e'/x] (\lambda y. e_1) &= \lambda y. [e'/x] e_1 \quad \text{if } x \in FV(e_1) \& y \notin FV(e'). \\
[e'/x] (\lambda y. e_1) &= \lambda z. [e'/x][z/y] e_1 \quad \text{if } x \in FV(e_1) \& y \in FV(e').
\end{align*}
\]
$\beta$ 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 $e_1 e \rightarrow_\beta e_2 e$, and $\lambda x.e_1 \rightarrow_\beta \lambda x.e_2$.

We write $e \rightarrow_\beta e'$ if $e \rightarrow_\beta \ldots \rightarrow_\beta e'$

$\equiv_\beta$ is the equivalence relation induced from $\rightarrow_\beta$
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 \( e \rightarrow^\beta e_1 \) and \( e \rightarrow^\beta e_2 \), then there exists \( e_3 \) such that \( e_1 \rightarrow^\beta e_3 \) and \( e_2 \rightarrow^\beta e_3 \)

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

\[
\begin{align*}
\text{(type)} & \quad A & ::= & \quad p \mid A_1 \rightarrow A_2 \\
\text{(term)} & \quad e & ::= & \quad x \mid \lambda x:A.e \mid e_1e_2
\end{align*}
\]
Simply typed lambda calculus

Type environment: \[ \Gamma ::= \cdot | \Gamma; x:A \]

Typing rules: \[ \Gamma \vdash e : A \quad \text{meaning under environment } \Gamma, \quad e \text{ is well-typed and has type } A. \]

\[
\begin{align*}
x & : A \vdash x : A \quad \text{(base)} \\
\Gamma, x : A & \vdash e : B \\
\Gamma & \vdash \lambda x : A. e : A \rightarrow B \quad \rightarrow_i \\
\Gamma & \vdash e_2 : A \quad \Delta \vdash e_1 : A \rightarrow B \\
\Gamma, \Delta & \vdash e_1 e_2 : B \quad \rightarrow_e
\end{align*}
\]
Simply typed lambda calculus

Another variation:

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

\[ \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B \rightarrow i} \]

\[ \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B \rightarrow e} \]

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


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: \(\land\), \(\lor\), \(\neg\), \(\bot\), ...... (and classic logic)
Propositional intuitionistic logic

\[ \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \land B} \quad ^\land i \]

\[ \frac{\Gamma \vdash A \land B}{\Gamma \vdash A} \quad ^\land e_1 \quad \frac{\Gamma \vdash A \land B}{\Gamma \vdash B} \quad ^\land e_2 \]

\[ \frac{\Gamma \vdash A}{\Gamma \vdash A \lor B} \quad ^\lor i_1 \]

\[ \frac{\Gamma \vdash B}{\Gamma \vdash A \lor B} \quad ^\lor i_2 \]

\[ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \quad ^\rightarrow i \]

\[ \frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B} \quad ^\rightarrow e \]

\[ \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \quad ^\neg i \]

\[ \frac{\Gamma \vdash \bot}{\Gamma \vdash \bot} \quad ^\bot e \]

no intro for \( \bot \)
A variation

basic sequent: \( \Gamma \vdash A \) if \( A \in \Gamma \)

- \( \Gamma \vdash A \) \( \Gamma \vdash B \) \( \Gamma \vdash A \land B \) \( \land_i \)
- \( \Gamma \vdash A \land B \) \( \Gamma \vdash A \) \( \land_{e_1} \)
- \( \Gamma \vdash A \land B \) \( \Gamma \vdash B \) \( \land_{e_2} \)
- \( \Gamma \vdash A \) \( \Gamma \vdash A \lor B \) \( \lor_{i_1} \)
- \( \Gamma \vdash A \lor B \) \( \Gamma, A \vdash C \) \( \Gamma, B \vdash C \) \( \lor_{e} \)
- \( \Gamma \vdash B \) \( \Gamma \vdash A \lor B \) \( \lor_{i_2} \)
- \( \Gamma, A \vdash B \) \( \Gamma \vdash A \rightarrow B \) \( \rightarrow_{i} \)
- \( \Gamma \vdash A \rightarrow B \) \( \Gamma \vdash B \) \( \rightarrow_{e} \)
- \( \Gamma, A \vdash \bot \) \( \Gamma \vdash \neg A \) \( \neg_{i} \)
- \( \Gamma \vdash \neg A \) \( \Gamma \vdash \bot \) \( \neg_{e} \)
- \( \Gamma \vdash \bot \) \( \bot_{e} \)

no intro for \( \bot \)
Extended typed lambda calculus

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

\[(\text{type}) \quad A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \neg A \]
\[
| \quad A_1 \land A_2 \mid A_1 \lor A_2
\]

\[(\text{term}) \quad e ::= x \mid \lambda x: A. e \mid e_1 e_2 \]
\[
| \quad \lambda^c x: A. e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e) \]
\[
| \quad (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \]
\[
| \quad \text{inj}_1(e) \mid \text{inj}_2(e) \]
\[
| \quad \text{case}(e, x_1.e_1, x_2.e_2)
\]
Extended typed lambda calculus

With more familiar notations:

(type) \[ A ::= p \mid \text{void} \mid A_1 \to A_2 \mid A \text{ cont} \]
\[ \mid A_1 \times A_2 \mid A_1 + A_2 \]

(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:**  \( \Gamma ::= \cdot | \Gamma, x : A \)

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

\[
\begin{align*}
\Gamma \vdash e_1 : A & \quad \Gamma \vdash e_2 : B & \quad \Gamma \vdash (e_1, e_2) : A \land B \\
\hline
\Gamma \vdash \pi_1(e) : A & \quad \Gamma \vdash \pi_2(e) : B \\
\hline
\Gamma \vdash e : A \land B & \quad \Gamma \vdash \pi_1(e) : A & \quad \Gamma \vdash \pi_2(e) : B
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : A & \quad \Gamma \vdash \text{inj}_1(e) : A \lor B & \quad \Gamma \vdash \text{inj}_2(e) : A \lor B \\
\hline
\Gamma \vdash e : B & \quad \Gamma \vdash \text{inj}_1(e) : A \lor B & \quad \Gamma \vdash \text{inj}_2(e) : A \lor B
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : A \lor B & \quad \Gamma, x_1 : A \vdash e_1 : C & \quad \Gamma, x_2 : B \vdash e_2 : C \\
\hline
\Gamma \vdash \text{case}(e, x_1.e_1, x_2.e_2) : C
\end{align*}
\]
Selected typing rules

\[ \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow i \]

\[ \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 \; e_2 : B} \rightarrow e \]

\[ \frac{\Gamma, x : A \vdash e : \bot}{\Gamma \vdash \lambda^c x : A. e : \neg A} \neg i \]

\[ \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : \neg A}{\Gamma \vdash \text{throw}(e_1, e_2) : \bot} \neg e \]

\[ \text{no intro for } \bot \]

\[ \frac{\Gamma \vdash e : \bot}{\Gamma \vdash \text{cast}(e) : A} \bot e \]
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, \ldots \}

- $O$ is a set of operators for putting them together:

  - **and** $\land$ (arity 2)
  - **or** $\lor$ (arity 2)
  - **implies** $\rightarrow$ (arity 2)
  - **not** $\neg$ (arity 1)
  - **falsity** $\bot$ (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, ...
- functions: +, -, *, ÷, ...
- predicates: ≤, ...
- formulas: x≤y+1, ...
- quantifiers: ∀, ∃
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 $f(t_1, t_2, \ldots, t_n)$ where $f \in \mathcal{F}$, and $t_1, t_2, \ldots, t_n$ are also terms
Write down its grammar:

\[(\text{terms}) \quad t ::= x \mid c \mid f(t_1, \ldots, t_n)\]
where \(x\) is a variable, \(c \in C\), & \(f \in F\) has arity \(n\).

We often merge \(C\) and \(F\) by treating all constants as 0-arity functions.

So a predicate vocabulary is just a pair \((F, P)\)
Predicate languages: formulas

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

- \(P(t_1, t_2, \ldots, t_n)\) where \(P \in \mathcal{P}\), and \(t_1, \ldots, t_n\) are terms over \(\mathcal{F}\), and the arity of \(P\) is \(n (n \geq 1)\).

- \(\bot\) is a formula.

- If \(A\) is a formula, then so is \(\neg A\).

- If \(A_1\) and \(A_2\) are formulas, then so are \((A_1 \land A_2)\), \((A_1 \lor A_2)\), \((A_1 \rightarrow A_2)\).

- If \(A\) is a formula, \(x\) is a variable, then so are \((\forall x. A)\) and \((\exists x. A)\).
Predicate languages

A summary:

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

(formulas) \( A ::= P(t_1,\ldots,t_n) \mid \bot \mid \neg A \mid A_1 \land A_2 \mid A_1 \lor A_2 \mid A_1 \rightarrow A_2 \mid \forall x.A \mid \exists x.A \) 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 !!!
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).
Natural deduction

\[
\frac{A}{A \land B} \quad \frac{B}{A \land B} \quad \land \text{intro}
\]
\[
\frac{A \land B}{A} \quad \frac{A \land B}{B} \quad \land \text{elim 1}
\]
\[
\frac{A \lor B}{A \lor B} \quad \text{vi}_1
\]
\[
\frac{A \lor B}{B} \quad \text{vi}_2
\]
\[
\frac{A \lor B}{C} \quad \frac{A}{C} \quad \frac{B}{C} \quad \lor \text{elim}
\]

\[
\frac{\vdots}{\neg A} \quad \neg \text{intro}
\]
\[
\frac{A}{\bot} \quad \neg \text{elim}
\]

no intro rule for \( \bot \)

\[
\frac{\bot}{A} \quad \bot \text{elim}
\]
Natural deduction

New rules for the two quantifiers:

\[
\begin{array}{ll}
\forall i & \frac{y \quad \vdots \quad A(y)}{\forall x. A(x)} \\
& \forall i \\
\forall e & \frac{\forall x. A(x)}{A(t)} \\
& \forall e
\end{array}
\]

\[
\begin{array}{ll}
\exists i & \frac{A(t)}{\exists x. A(x)} \\
& \exists i \\
\exists e & \frac{\exists x. A(x)}{B} \\
& \exists e
\end{array}
\]

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 \( A \in \Gamma \)

\[
\begin{align*}
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} & \quad \landi \\
\frac{\Gamma \vdash A \land B}{\Gamma \vdash A} & \quad \lande_1 \\
\frac{\Gamma \vdash A \land B}{\Gamma \vdash B} & \quad \lande_2 \\
\frac{\Gamma \vdash A}{\Gamma \vdash A \lor B} & \quad \lori_1 \\
\frac{\Gamma \vdash B}{\Gamma \vdash A \lor B} & \quad \lori_2 \\
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} & \quad \rightarrowi \\
\frac{\Gamma \vdash A \quad \Gamma \vdash A \rightarrow B}{\Gamma \vdash B} & \quad \rightarrowe \\
\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} & \quad \negi \\
\frac{\Gamma \vdash A}{\Gamma \vdash \bot} & \quad \nega \\
\frac{\Gamma \vdash \bot}{\Gamma \vdash \bot} & \quad \bote
\end{align*}
\]

no intro for \( \bot \)
Natural deduction in sequent style

\[
\begin{align*}
x \in \Gamma & \quad \frac{x \in \Gamma}{\Gamma \vdash x} \\
\frac{\Gamma \vdash t_i \ (1 \leq i \leq n)}{\frac{f \in \mathcal{F}}{\Gamma \vdash f(t_1, \ldots, t_n)}} \\
\end{align*}
\]

\[
\begin{align*}
\frac{\Gamma, y \vdash A(y)}{\Gamma \vdash \forall x. A(x)} & \quad \forall_i \\
\frac{\Gamma \vdash \forall x. A(x)}{\frac{\Gamma \vdash t}{\Gamma \vdash A(t)}} & \quad \forall_e \\
\frac{\Gamma \vdash t \quad \Gamma \vdash A(t)}{\Gamma \vdash \exists x. A(x)} & \quad \exists_i \\
\frac{\Gamma \vdash \exists x. A(x) \quad \Gamma, y, A(y) \vdash B}{\frac{\Gamma \vdash B}{\exists_e}} \\
\end{align*}
\]
Extended typed $\lambda$-calculus revisited

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

(type) \[ A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \neg A \]
\[ \mid A_1 \land A_2 \mid A_1 \lor A_2 \]

(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) $A ::= P(t_1, ..., t_n) \mid \bot \mid A_1 \to A_2$
$\mid \neg A \mid A_1 \land A_2 \mid A_1 \lor A_2 \mid \forall x.A \mid \exists x.A$

(exp) $e ::= z \mid \lambda z : A.e \mid e_1e_2$
$\mid \lambda^c z : A.e \mid \text{throw}(e_1, e_2) \mid \text{cast}(e)$
$\mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e)$
$\mid \text{inj}_1(e) \mid \text{inj}_2(e) \mid \text{case}(e, z_1.e_1, z_2.e_2)$
$\mid \lambda x.e \mid e[\tau] \mid \text{pack}(\tau, e:A(\tau))$
$\mid \text{open } e_1 \text{ as } (x, z) \text{ in } e_2$

(term) $t ::= x \mid f(t_1, ..., t_n)$
Selected typing rules

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

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

\[
\begin{align*}
\Gamma \vdash e_1 : A & \quad \Gamma \vdash e_2 : B \\
\hline
\Gamma \vdash (e_1, e_2) : A \land B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash e : A \land B & \quad \text{\texttt{\&e}_1} \\
\hline
\Gamma \vdash \pi_1(e) : A
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash e : A \land B & \quad \text{\texttt{\&e}_2} \\
\hline
\Gamma \vdash \pi_2(e) : B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash e : A & \quad \text{\texttt{\lori}_1} \\
\hline
\Gamma \vdash \text{inj}_1(e) : A \lor B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash e : B & \quad \text{\texttt{\lori}_1} \\
\hline
\Gamma \vdash \text{inj}_2(e) : A \lor B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash e_1 : A \lor B & \quad \Gamma, z_1 : A \vdash e_1 : C \\
\hline
\Gamma, z_2 : B \vdash e_2 : C \\
\hline
\Gamma \vdash \text{case}(e, z_1.e_1, z_2.e_2) : C
\end{align*}
\]

\texttt{\&e}_1, \texttt{\&e}_2, \texttt{\lori}_1, \texttt{\lori}_1, \texttt{\lori}_1
Selected typing rules

\[ \frac{\Gamma, z : A \vdash B}{\Gamma \vdash \lambda z : A.e : A \rightarrow B} \rightarrow i \]

\[ \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 \ e_2 : B} \rightarrow e \]

\[ \frac{\Gamma, z : A \vdash e : \bot}{\Gamma \vdash \lambda^c z : A.e : \neg A} \neg i \]

\[ \frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : \neg A}{\Gamma \vdash \text{throw}(e_1, e_2) : \bot} \neg e \]

no intro for \( \bot \)

\[ \frac{\Gamma \vdash e : \bot}{\Gamma \vdash \text{cast}(e) : A} \bot e \]

\[ \Gamma \vdash e : \bot \]

\[ \Gamma \vdash \text{cast}(e) : A \]
Here are the typing rules for terms, ∀ and ∃

\[
\begin{align*}
\Gamma \vdash x & \quad \frac{x \in \Gamma}{\Gamma \vdash x} & \frac{\Gamma \vdash t_i \quad (1 \leq i \leq n)}{\Gamma \vdash f(t_1, \ldots, t_n)} & \frac{f \in \mathcal{F}}{f \text{ has arity } n} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x \vdash e : A(x) & \quad \frac{\Gamma \vdash e : \forall x.A(x)}{\Gamma \vdash \lambda x.e : \forall x.A(x)} & \quad \frac{\Gamma \vdash e[t] : A(t)}{\Gamma \vdash t} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t & \quad \frac{\Gamma \vdash e : A(t)}{\Gamma \vdash \text{pack}(t, e : A(t)) : \exists x.A(x)} & \frac{\Gamma \vdash e_1 : \exists x.A(x)}{\Gamma \vdash \text{open } e_1 \text{ as } (y, z) \text{ in } e_2 : B} \\
\end{align*}
\]

∀e

∀i

∃i

∃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., \(\bot, \neg, \land, \lor, \rightarrow\)), each with a specified arity.
- \(\mathcal{Q}\) is a set of quantifiers (e.g., \(\forall, \exists\)).
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 $M$.
- $F = (f_{o_1}, f_{o_2}, \ldots, f_{o_n})$ is a set of functions, one for each operator in $O = \{o_1, o_2, \ldots, o_n\}$, 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
Semantics of predicate logic

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

The following valuation system for this gives classical logic:

- \( M = \{ t, f \} \) and \( D = \{ t \} \)
- \( F \) is defined as in the propositional case
- \( G = \{ g_\forall, g_\exists \} \) where
  - \( g_\forall(X) = f \) if \( f \in X \)
  - \( g_\forall(X) = t \) otherwise
  - \( g_\exists(X) = t \) if \( t \in X \)
  - \( g_\exists(X) = f \) otherwise
Semantics of predicate logic

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

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

Essentially, for each \(f \in F\) with arity \(n\), \(a(f)\) is a function from the set \(I^n\) to the set \(I\) (i.e., \(a(f) : I^n \to I\)).
Semantics of predicate logic

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

- If \(p \in P\) has arity \(n\) and \(t_1, \ldots, t_n \in T\) then
  \[ v_a(p(t_1, \ldots, t_n)) = a(p)(a(t_1), \ldots, a(t_n)). \]

- \(v_a(o(A_1, \ldots, A_p)) = f_o(v_a(A_1), \ldots, v_a(A_n))\) where \(n\) is the arity of \(o\).

- \(v_a(q \times A) = g_q(\{v_{a'}(A) \mid a' = a[x \mapsto i] \text{ for all } i \in I\})\)

An interpretation is a valuation system plus an assignment.
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 $f^\mathcal{M} : I^n \to I$ 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 $A$. 

Models [Huth & Ryan]
An environment \( l \) is a mapping from every variable \( x \) (in \( V \)) to a value \( l(x) \) of \( I \).

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

From \( v_a \), we can define a semantic entailment relation \( \Gamma \vdash_{v_a} A \).

We write \( \Gamma \vdash A \) if \( \Gamma \vdash_{v_a} A \) is true for all \((a, I)\) (or for all \((M, I)\) ).
Important properties

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

A presentation $\vdash$ is **complete with respect to a semantics** $\vdash$ if for all $\Gamma$ and $A$, if $\Gamma \vDash A$, 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 $\emptyset \vdash \neg A$. 
A presentation $\vdash$ is **absolute consistent** if there exists an $A$ such that $\emptyset \not\models A$.

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 $\Gamma \models A$ or $\Gamma \models \neg A$.

[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\textsuperscript{st} order, 2\textsuperscript{nd} order, ... higher-order)

- Typed $\lambda$-calculi ($\lambda \rightarrow$, System F, $F_\omega$)

- Predicate logic (1\textsuperscript{st} order, 2\textsuperscript{nd} order, ... higher-order)

- Many-sorted predicate logic

- Dependently typed $\lambda$-calculi
A propositional language $L = (P, O)$:

- $P$ is a set of atomic sentences:
  $$\{p, q, r, \ldots\}$$

- $O$ is a set of operators for putting them together:

  - **and** $\land$ (arity 2)
  - **or** $\lor$ (arity 2)
  - **implies** $\rightarrow$ (arity 2)
  - **not** $\neg$ (arity 1)
  - **falsity** $\bot$ (arity 0)
First-order propositional logic

Natural deduction rules:

\[ \frac{A \quad \vdots \quad B}{A \rightarrow B} \rightarrow i \]

\[ \frac{A}{A \rightarrow B \quad B} \rightarrow e \]

no intro rule for \( \bot \)

\[ \frac{\bot}{A} \rightarrow e \]
First-order propositional logic

Natural deduction rules in sequent style:

basic sequent: \( \Gamma \vdash A \) if \( A \in \Gamma \)

\[ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow i \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \rightarrow B} \rightarrow e \]

no intro for \( \bot \)

\[ \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot e \]
Simply typed lambda calculus

Syntax for the simply typed \( \lambda \) calculus:

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

(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: $\Gamma ::= \cdot | \Gamma, x : A$

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

$\Gamma, x : A \vdash e : B \rightarrow i$

$\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B \rightarrow e$

no intro for $\bot$

$\Gamma \vdash e : \bot \rightarrow e$

$\Gamma \vdash \text{cast}(e) : A \bot e$
Second order propositional language

We add universal quantification over atomic sentences:

\[(\text{props}) \quad A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \forall p.A\]

\[
\begin{array}{c}
p \\
\vdots \\
A(p) \\
\hline
\forall p. A(p)
\end{array}
\]

\[
\begin{array}{c}
A \rightarrow A' \\
A(p) \\
\hline
\forall p. A(p) \\
\forall A'(A(A'))
\end{array}
\]

\[
\begin{array}{c}
A(p) \\
\forall p. A(p) \\
\hline
\forall A'. A(A') \\
A
\end{array}
\]

\[
\begin{array}{c}
\forall p. A(p) \\
\forall A' \\
\hline
\forall p. A(p) \\
\forall A'. A(A') \\
\forall A'
\end{array}
\]
Polymorphically typed lambda calculus

Syntax of system F (a.k.a., 2nd order polymorphically typed \( \lambda \) calculus):

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

(term) \( e ::= x \mid \lambda x: A. e \mid e_1 e_2 \)
\( \mid \text{cast}(e) \)
\( \mid \lambda p. e \mid e(A) \)
Polymorphically typed lambda calculus

No need for \(\bot\) 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:  \( \Gamma ::= \cdot | \Gamma, x : A | \Gamma, p \)

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

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \rightarrow B}
\]

\[
\frac{\Gamma, p \vdash A}{\Gamma \vdash \forall p. A}
\]

\[
\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \quad \text{i}
\]

\[
\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 \ e_2 : B} \quad \text{e}
\]

\[
\frac{\Gamma, p \vdash e : A}{\Gamma \vdash \lambda p. e : \forall p. A} \quad \text{vi}
\]

\[
\frac{\Gamma \vdash e : \forall p. A \quad \Gamma \vdash A'}{\Gamma \vdash e \ [A'] : [A'/p]A} \quad \text{ve}
\]
System F with kind annotations

We can make up a “kind” for all types

(kind) \( K ::= \Omega \)

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

(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 $p \in \Gamma$

$$
\begin{align*}
\Gamma \vdash A : \Omega & \quad \Gamma \vdash B : \Omega \\
\Gamma & \vdash A \rightarrow B : \Omega
\end{align*}
$$

$$
\Gamma, p : K \vdash A : \Omega
\quad \Gamma \vdash \forall p : K . A : \Omega
$$

$$
\Gamma, x : A \vdash e : B
\quad \Gamma \vdash \lambda x : A . e : A \rightarrow B \rightarrow i
$$

$$
\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B
\quad \Gamma \vdash e_1 \; e_2 : B \rightarrow e
$$

$$
\Gamma, p : K \vdash e : A
\quad \Gamma \vdash \lambda p : K . e : \forall p : K . A \forall i
$$

$$
\Gamma \vdash e : \forall p : K . A \quad \Gamma \vdash A' : K
\quad \Gamma \vdash e [A'] : [A'/p]A \forall e
$$
A variant: splitting the environment

kind environment: \( \Delta ::= \cdot \mid \Delta, p : K \)

type environment: \( \Gamma ::= \cdot \mid \Gamma, x : A \)

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

\[
\begin{align*}
\Delta \vdash A : \Omega & \quad \Delta \vdash B : \Omega \\
\frac{}{\Delta \vdash A \rightarrow B : \Omega} \\
\Delta, \Gamma, x : A & \vdash B \\
\frac{}{\Delta ; \Gamma \vdash \lambda x : A.e : A \rightarrow B \rightarrow i} \\
\Delta, p : K & \vdash e : A \\
\frac{}{\Delta ; \Gamma \vdash \lambda p : K.e : \forall p : K.A \forall i} \\
\Delta, \Gamma & \vdash e : \forall p : K.A \\
\frac{}{\Delta ; \Gamma \vdash e \left[ A' \right] : \left[ A'/p \right] A \forall e} \\
\end{align*}
\]

\[
\begin{align*}
\Delta, p : K & \vdash A : \Omega \\
\frac{}{\Delta \vdash \forall p : K.A : \Omega} \\
\Delta, \Gamma & \vdash e_2 : A \\
\frac{}{\Delta ; \Gamma \vdash e_1 e_2 : B \rightarrow e} \\
\Delta ; \Gamma & \vdash e_1 : A \rightarrow B \\
\frac{}{\Delta ; \Gamma \vdash e_1 e_2 : B \rightarrow e} \\
\end{align*}
\]
We can also add higher-order predicates:

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

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

\[
\begin{array}{c}
p : K \\
\vdash A(p)
\end{array}
\] \( \forall_i \)

\[
\forall p : K. A(p) \quad A' : K \\
\frac{A(A')}{A'}
\] \( \forall_e \)
Higher-order polymorphic $\lambda$-calculus

Add type-functions to system $F$, we get $F_\omega$

(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 $p \in \Gamma$

\[
\frac{\Gamma \vdash A : \Omega \quad \Gamma \vdash B : \Omega}{\Gamma \vdash A \rightarrow B : \Omega}
\quad
\frac{\Gamma, p : K \vdash A : \Omega}{\Gamma \vdash \forall p : K.A : \Omega}
\]

\[
\frac{\Gamma, p : K_1 \vdash A : K_2}{\Gamma \vdash \lambda p : K_1.A : K_1 \rightarrow K_2}
\quad
\frac{\Gamma \vdash A : K_1 \rightarrow K_2 \quad \Gamma \vdash B : K_1}{\Gamma \vdash A(B) : K_2}
\]

\[
\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A.e : A \rightarrow B} \quad
\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \quad
\frac{\Gamma \vdash e_2 : A \quad \Gamma \vdash e_1 : A \rightarrow B}{\Gamma \vdash e_1 e_2 : B} \quad
\]

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

\[
\frac{\Gamma \vdash e : \forall p : K.A \quad \Gamma \vdash A' : K}{\Gamma \vdash e[A'] : [A'/p]A} \quad
\frac{\Gamma \vdash e : \forall p : K.A \quad \Gamma \vdash A' : K}{\Gamma \vdash e[A'] : [A'/p]A}
\]
First-order predicate languages

A simplified version:

(terms) \[ t ::= x \mid f(t_1, \ldots, t_n) \quad \text{where } f \in \mathcal{F} \]

(formulas) \[ A ::= P(t_1, \ldots, t_n) \mid \bot \mid A_1 \rightarrow A_2 \]
\[ \mid \forall x. A \quad \text{where } P \in \mathcal{P} \]
Natural deduction

New rules for the two quantifiers:

\[
\begin{align*}
&\frac{y}{\forall x. A(x)} \quad \forall i \\
&\frac{A(y)}{\forall x. A(x)} \quad \forall e \\
&\frac{\exists x. A(x)}{A(t)} \quad \exists i \\
&\frac{A(t)}{\exists x. A(x)} \quad \exists e \\
&\frac{\exists x. A(x)}{\forall x. A(x)} \quad \forall e
\end{align*}
\]

Note: In \(\forall i\) and \(\exists e\), variable \(y\) doesn't occur free outside the box.
Natural deduction in sequent style

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

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

\[
\frac{\Gamma \vdash t \quad \Gamma \vdash A(t)}{\Gamma \vdash \exists x. A(x)} \quad \exists i \\
\frac{\Gamma \vdash \exists x. A(x)}{\Gamma \vdash B} \quad \exists e
\]

\[
\Gamma, y, A(y) \vdash B
\]
A minimized version of $\lambda$Pred:

(type) \[ A ::= P(t_1,\ldots,t_n) \mid \bot \]
\[ \mid A_1 \to A_2 \mid \forall x.A \]

(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) \[ t ::= x \mid f(t_1,\ldots,t_n) \]
Higher-order predicate languages

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

(terms) \( t ::= x \mid f(t_1, \ldots, t_n) \mid \lambda x:T.t \mid t_1(t_2) \)

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

(formulas) \( A ::= P(t_1, \ldots, t_n) \mid A_1 \to A_2 \mid \forall x:T.A \mid p \mid \forall p:K.A \mid \lambda p:K.A \mid A_1(A_2) \)

(prf-terms) \( e ::= z \mid \lambda z:A.e \mid e_1 e_2 \mid \lambda x:T.e \mid e(t) \mid \lambda p:K.e \mid e(A) \)
We’ll add back function and predicate symbols via inductive definitions later:

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

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

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

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

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

(prf-terms)  \( e ::= z | \lambda z:A.e | e_1e_2 | \lambda x:T.e | e(t) \\
\qquad | \lambda p:K.e | e(A) \)
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)  \( e ::= z \quad | \quad \lambda z:A.e \quad | \quad e_1e_2 \quad | \quad \lambda x:T.e \quad | \quad e(t) \quad | \quad \lambda p:K.e \quad | \quad e(A) \)
Unifying →, ∀, and Π

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

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

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

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

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

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

\[(tm-knd-scm) \quad u ::= \, \text{Type}'\]

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

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

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

\[(p-knd-scm) \quad w ::= \, \text{Type}\]

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

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

\[(prf-terms) \quad e ::= \, z \quad | \quad \lambda z : A.e \quad | \quad e_1 e_2 \quad | \quad \lambda x : T.e \quad | \quad e(t) \quad | \quad \lambda p : K.e \quad | \quad e(A)\]
The big merge

\[(\text{tm,p-knd-scm}) \ u, w ::= \text{Type'} | \text{Type}\]

\[(\text{tm,p-kind}) \ U, K ::= \text{Set} | \Pi\alpha:U_1.U_2\]

\[\quad | \text{Prop} | \Pi p:K_1.K_2 | \Pi x:T.K\]

\[(\text{tm-types,form}) \ T, A ::= \alpha | \Pi x:T_1.T_2 | \Pi\alpha:U.T | \lambda\alpha:U.T | T_1(T_2)\]

\[\quad | p | \Pi z:A_1.A_2 | \lambda x:T.A | A(t)\]

\[\quad | \Pi x:T.A | p | \Pi p:K.A | \lambda p:K.A | A_1(A_2)\]

\[(\text{terms,prf-tm}) \ t, e ::= x | \lambda x:T.t | t_1(t_2) | \lambda\alpha:U.t | t(T)\]

\[\quad | z | \lambda z:A.e | e_1e_2 | \lambda x:T.e | e(t)\]

\[\quad | \lambda p:K.e | e(A)\]
Calculus of constructions

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

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

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

(term) \( e ::= x \\
| \lambda x:A.e | e_1e_2 \\
| \lambda p:K.e | e(A) \)

With two sorts:

- Type
- Prop

Four \( \Pi \)'s:

- \((\text{Prop},\text{Prop})\)
- \((\text{Prop},\text{Type})\)
- \((\text{Type},\text{Prop})\)
- \((\text{Type},\text{Type})\)
Pure type systems

(T :: s | x | Πx:T₁. T₂
  | λx:T₁. T₂ | T₁(T₂)

- To rule out ill-formed terms, we define what are well-formed Π’s
- A set of sorts, e.g., Type, Prop
- A set of rules \((s₁, s₂, s₃)\): if \(T₁\) has sort \(s₁\), and \(T₂\) has sort \(s₂\), then \(Πx:T₁. T₂\) has sort \(s₃\)
- Write \((s₁, s₂)\) to mean \((s₁, s₂, s₂)\)

CC with two sorts:

- Type
- Prop

Four Π’s:
- (Prop, Prop)
- (Prop, Type)
- (Type, Prop)
- (Type, Type)
CC with four sorts:

\[(ptm)\]

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

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 three sorts

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

Nine \( \Pi \)'s:

- \((\text{Set}, \text{Set})\)
- \((\text{Type}, \text{Set})\)
- \((\text{Type}, \text{Type})\)
- \((\text{Prop}, \text{Prop})\)
- \((\text{Type}, \text{Prop})\)
- \((\text{Set}, \text{Type})\)
- \((\text{Set}, \text{Prop})\)
- \((\text{Prop}, \text{Set})\)
- \((\text{Prop}, \text{Type})\)

CC with three sorts:

- Type
- Prop
- Set
CC with infinite number of universes

\[(\text{ptm})\quad T ::= s \mid x \mid \Pi x:T_1.T_2 \mid \lambda x:T_1.T_2 \mid T_1(T_2)\]

Infinitely many \(\Pi\)'s:
- \((\text{Set},\text{Set})\), \((\text{TypeI},\text{Set})\)
- \((\text{Prop},\text{Prop})\), \((\text{TypeI},\text{Prop})\)
- \((\text{Set},\text{TypeI})\), \((\text{Set},\text{Prop})\)
- \((\text{Prop},\text{Set})\), \((\text{Prop},\text{TypeI})\)

\((\text{TypeI}, \text{TypeJ}, \text{TypeK})\) where \(I \leq K\) and \(J \leq K\)
Adding inductive definitions

<table>
<thead>
<tr>
<th>(tm-knd-scm)  u ::=  Type’</th>
</tr>
</thead>
<tbody>
<tr>
<td>(tm-kind)    U ::= Set</td>
</tr>
<tr>
<td>(tm-types)   T ::=  α</td>
</tr>
<tr>
<td>(terms)      t ::=  x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(p-knd-scm)  w ::=  Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(prop-kind)  K ::=  Prop</td>
</tr>
<tr>
<td>(formulas)   A ::=  p</td>
</tr>
<tr>
<td>(prf-terms)  e ::=  z</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
CC with inductive definitions

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

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

(type) \( A ::= p \mid \prod_{x:A_1.A_2} \mid \prod_{p:K.A} \mid \lambda_{p:K.A} A_1(A_2) \mid \lambda_{x:A_1.A_2} A(e) \mid \text{Ind}(p:K)(A_1,\ldots,A_n) \)

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

With two sorts:

Type

Prop

Four \( \Pi \)'s:

(Prop,Prop)

(Prop,Type)

(Type,Prop)

(Type,Type)
Inductive definitions: syntactic sugar

Inductive $I : K = C_1 : A_1$

| $C_2 : A_2$
| $\ldots \ldots$

| $C_n : A_n$

represents:

$I \equiv \text{Ind}(p : K)([p/I]A_1, \ldots, [I/p]A_n)$

$C_1 \equiv \text{Ctor}(1, I)$

$C_2 \equiv \text{Ctor}(2, I)$

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

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