<table>
<thead>
<tr>
<th>Logic</th>
<th>$\leftrightarrow$</th>
<th>Type system</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions $\phi$</td>
<td>$\leftrightarrow$</td>
<td>types $\tau$</td>
</tr>
<tr>
<td>proofs $p$</td>
<td>$\leftrightarrow$</td>
<td>expressions $M$</td>
</tr>
<tr>
<td>‘$p$ is a proof of $\phi$’</td>
<td>$\leftrightarrow$</td>
<td>‘$M$ is an expression of type $\tau$’</td>
</tr>
<tr>
<td>simplification of proofs</td>
<td>$\leftrightarrow$</td>
<td>reduction of expressions</td>
</tr>
</tbody>
</table>
Curry-Howard correspondence

Logic $\leftrightarrow$ Type system

- Propositions $\phi$ $\leftrightarrow$ Types $\tau$
- Proofs $p$ $\leftrightarrow$ Expressions $M$
- ‘$p$ is a proof of $\phi$’ $\leftrightarrow$ ‘$M$ is an expression of type $\tau$’

Girard’s Linear Logic $\rightarrow$ Usage analysis in compilers
Linear implication

\[ \frac{\Gamma, \psi \vdash \psi}{\Gamma \vdash \psi} \quad \frac{\Gamma \vdash \phi \rightarrow \psi}{\Gamma, \Delta \vdash \psi} \quad (\Gamma \cap \Delta = \emptyset) \]

Linear conjunction (tensor)

\[ \frac{\Gamma \vdash \psi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \psi \otimes \psi} \quad (\Gamma \cap \Delta = \emptyset) \]

\[ \frac{\Gamma \vdash \psi \otimes \psi \quad \Delta, \psi, \psi \vdash \Theta}{\Gamma, \Delta \vdash \Theta} \quad (\Gamma \cap \Delta = \emptyset) \]
Curry-Howard correspondence

Applications

Logic $\leftrightarrow$ Type system

propositions $\phi$ $\leftrightarrow$ types $\tau$

proofs $p$ $\leftrightarrow$ expressions $M$

‘$p$ is a proof of $\phi$’ $\leftrightarrow$ ‘$M$ is an expression of type $\tau$’

simplification of proofs $\leftrightarrow$ reduction of expressions

Linear Temporal Logic $\sim$ functional reactive programming

Modal logics $\sim$ partial evaluation & run-time code generation
Type-inference versus proof search

*Type-inference*: given $\Gamma$ and $M$, is there a type $\tau$ such that $\Gamma \vdash M : \tau$?

(For PLC/2IPC this is decidable.)

*Proof-search*: given $\Gamma$ and $\phi$, is there a proof term $M$ such that $\Gamma \vdash M : \phi$?

(For PLC/2IPC this is undecidable.)
Curry-Howard correspondence

<table>
<thead>
<tr>
<th>Logic</th>
<th>↔</th>
<th>Type system</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions $\phi$</td>
<td>↔</td>
<td>types $\tau$</td>
</tr>
<tr>
<td>proofs $p$</td>
<td>↔</td>
<td>expressions $M$</td>
</tr>
<tr>
<td>‘$p$ is a proof of $\phi$’</td>
<td>↔</td>
<td>‘$M$ is an expression of type $\tau$’</td>
</tr>
<tr>
<td>simplification of proofs</td>
<td>↔</td>
<td>reduction of expressions</td>
</tr>
</tbody>
</table>

Applications

- proof assistants
  - Agda
  - Coq

- dependently typed $\lambda$-calculus
  - (e.g. Calculus of Constructions)
Curry-Howard correspondence

<table>
<thead>
<tr>
<th>Logic</th>
<th>⇔</th>
<th>Type system</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions $\phi$</td>
<td>⇔</td>
<td>types $\tau$</td>
</tr>
<tr>
<td>proofs $p$</td>
<td>⇔</td>
<td>expressions $M$</td>
</tr>
<tr>
<td>‘$p$ is a proof of $\phi$’</td>
<td>⇔</td>
<td>‘$M$ is an expression of type $\tau$’</td>
</tr>
</tbody>
</table>

simplification of proofs ⇔ reduction of expressions

E.g. 2IPC ⇔ PLC

C-H also applied to predicate logic
Curry-Howard correspondence

higher-order
intuitionistic
logic

Predicate

Calciums
of
Constructions
Pure Type Systems – typing rules

(axiom) \[ \Diamond \vdash s_1 : s_2 \quad \text{if} \quad (s_1, s_2) \in \mathcal{A} \]

(start) \[ \Gamma \vdash A : s \quad \Gamma, x : A \vdash x : A \quad \text{if} \quad x \not\in \text{dom}(\Gamma) \]

(weaken) \[ \Gamma \vdash M : A \quad \Gamma \vdash B : s \quad \Gamma, x : B \vdash M : A \quad \text{if} \quad x \not\in \text{dom}(\Gamma) \]

(conv) \[ \Gamma \vdash M : A \quad \Gamma \vdash B : s \quad \Gamma \vdash M : B \quad \text{if} \quad A =_\beta B \]

(prod) \[ \Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma \vdash \Pi x : A (B) : s_3 \quad \text{if} \quad (s_1, s_2, s_3) \in \mathcal{R} \]

(abs) \[ \Gamma, x : A \vdash M : B \quad \Gamma \vdash \Pi x : A (B) : s \quad \Gamma \vdash \lambda x : A (M) : \Pi x : A (B) \]

(app) \[ \Gamma \vdash M : \Pi x : A (B) \quad \Gamma \vdash N : A \quad \Gamma \vdash MN : B[N/x] \]

(A, B, M, N range over pseudoterms, s, s_1, s_2, s_3 over sort symbols)
Calculus of Constructions

is the Pure Type System $\lambda C$, where $C = (S_C, A_C, R_C)$ is the PTS specification with

- $S_C \triangleright\{\text{Prop}, \text{Set}\}$ (Prop = a sort of propositions, Set = a sort of types)
- $A_C \triangleright\{(\text{Prop}, \text{Set})\}$ (Prop is one of the types)
- $R_C \triangleright\{(\text{Prop}, \text{Prop}, \text{Prop}), (\text{Set}, \text{Prop}, \text{Prop}),$
  - $(\text{Prop}, \text{Set}, \text{Set}), (\text{Set}, \text{Set}, \text{Set})\}$
Calculus of Constructions

is the Pure Type System $\lambda C$, where $C = (\mathcal{S}_C, \mathcal{A}_C, \mathcal{R}_C)$ is the PTS specification with

\[
\begin{align*}
\mathcal{S}_C & \triangleq \{ \text{Prop}, \text{Set} \} \\
\mathcal{A}_C & \triangleq \{ (\text{Prop}, \text{Set}) \} \\
\mathcal{R}_C & \triangleq \{ (\text{Prop}, \text{Prop}, \text{Prop})^1, (\text{Set}, \text{Prop}, \text{Prop}), \\
& \quad \quad \quad (\text{Prop}, \text{Set}, \text{Set}), (\text{Set}, \text{Set}, \text{Set}) \} \\
\end{align*}
\]

1. Prop has implications, $\phi \rightarrow \psi = \Pi x : \phi (\psi)$ (where $\phi, \psi : \text{Prop}$ and $x \notin \text{fv}(q)$).
Calculus of Constructions

is the Pure Type System $\lambda C$, where $C = (S_C, A_C, R_C)$ is the PTS specification with

$$\begin{align*}
S_C & \triangleq \{\text{Prop, Set}\} \\
A_C & \triangleq \{(\text{Prop, Set})\} \\
R_C & \triangleq \{(\text{Prop, Prop, Prop})^1, (\text{Set, Prop, Prop})^2, \\
& \quad (\text{Prop, Set, Set}), (\text{Set, Set, Set})\}
\end{align*}$$

1. $\text{Prop}$ has implications, $\phi \to \psi = \Pi x : \phi (\psi)$ (where $\phi, \psi : \text{Prop}$ and $x \notin fv(q)$).

2. $\text{Prop}$ has universal quantifications over elements of a type, $\Pi x : A (\phi(x))$ (where $A : \text{Set}$ and $x : A \vdash \phi(x) : \text{Prop}$).

N.B. $A$ might be $\text{Prop}$ ($\lambda 2 \subseteq \lambda C$).
Calculus of Constructions

is the Pure Type System $\lambda C$, where $C = (S_C, A_C, R_C)$ is the
PTS specification with

\[
S_C \triangleq \{\text{Prop, Set}\} \\
A_C \triangleq \{(\text{Prop, Set})\} \\
R_C \triangleq \{(\text{Prop, Prop, Prop})^1, (\text{Set, Prop, Prop})^2, \\
(\text{Prop, Set, Set}), (\text{Set, Set, Set})^4\}
\]

1. Prop has implications, $\phi \rightarrow \psi = \Pi x : \phi (\psi)$ (where $\phi, \psi : \text{Prop}$ and $x \notin \text{fv}(q)$).

2. Prop has universal quantifications over elements of a type, $\Pi x : A (\phi(x))$
   (where $A : \text{Set}$ and $x : A \vdash \phi(x) : \text{Prop}$).
   N.B. $A$ might be Prop ($\lambda 2 \subseteq \lambda C$).

4. Set has dependent function types, $\Pi x : A (B(x))$ (where $A : \text{Set}$ and $x : A \vdash B(x) : \text{Set}$).
Calculus of Constructions

is the Pure Type System $\lambda C$, where $C = (S_C, A_C, R_C)$ is the PTS specification with

$$S_C \triangleq \{\text{Prop, Set}\}$$
$$A_C \triangleq \{ (\text{Prop, Set}) \}$$
$$R_C \triangleq \{ (\text{Prop, Prop, Prop})^1, (\text{Set, Prop, Prop})^2, (\text{Prop, Set, Set})^3, (\text{Set, Set, Set})^4 \}$$

1. Prop has implications, $\phi \rightarrow \psi = \Pi x : \phi (\psi)$ (where $\phi, \psi : \text{Prop}$ and $x \notin fv(\psi)$).

2. Prop has universal quantifications over elements of a type, $\Pi x : A (\phi(x))$ (where $A : \text{Set}$ and $x : A \vdash \phi(x) : \text{Prop}$).
   N.B. $A$ might be $\text{Prop}$ ($\lambda 2 \subseteq \lambda C$).

3. Set has types of function dependent on proofs of a proposition, $\Pi x : p (A(x))$ (where $p : \text{Prop}$ and $x : p \vdash A(x) : \text{Set}$).

4. Set has dependent function types, $\Pi x : A (B(x))$ (where $A : \text{Set}$ and $x : A \vdash B(x) : \text{Set}$).
Some general properties of $\lambda C$

- It extends both $\lambda 2$ (PLC) and $\lambda \omega$ ($F_\omega$).
Some general properties of $\lambda C$

- It extends both $\lambda 2$ (PLC) and $\lambda \omega$ ($F_\omega$).
- $\lambda C$ is strongly normalizing.
- Type-checking and typeability are decidable.
Logical operations definable in 2IPC

- **Truth** \( \top \triangleq \forall p \ (p \rightarrow p) \)
- **Falsity** \( \bot \triangleq \forall p \ (p) \)
- **Conjunction** \( \phi \land \psi \triangleq \forall p \ ((\phi \rightarrow \psi \rightarrow p) \rightarrow p) \) (where \( p \notin \text{fv}(\phi, \psi) \))
- **Disjunction** \( \phi \lor \psi \triangleq \forall p \ ((\phi \rightarrow p) \rightarrow (\psi \rightarrow p) \rightarrow p) \) (where \( p \notin \text{fv}(\phi, \psi) \))
- **Negation** \( \neg \phi \triangleq \phi \rightarrow \bot \)
- **Bi-implication** \( \phi \leftrightarrow \psi \triangleq (\phi \rightarrow \psi) \land (\psi \rightarrow \phi) \)

\[
\begin{align*}
p \rightarrow q & \triangleq \forall x : p \ (q) \quad x \notin \text{fv}(p) \\
\forall p (\psi) & \triangleq \forall p : \text{Prop} \ (\psi)
\end{align*}
\]
Some general properties of $\lambda C$

- It extends both $\lambda 2$ (PLC) and $\lambda \omega$ ($F_\omega$).

- $\lambda C$ is strongly normalizing.

- Type-checking and typeability are decidable.

- $\lambda C$ is logically consistent (relative to the usual foundations of classical mathematics), that is, there is no pseudo-term $t$ satisfying $\Box \vdash t : \Pi p : \text{Prop} (p)$.

  Indeed there is no proof of LEM ($\Pi p : \text{Prop} (\neg p \vee p)$).
Leibniz equality in $\lambda C$

Gottfried Wilhelm Leibniz (1646–1716),
identity of indiscernibles:
\textit{duo quaedam communes proprietates eorum nequaquam possit}
(two distinct things cannot have all their properties in common).
Leibniz equality in $\lambda C$

Gottfried Wilhelm Leibniz (1646–1716),

**identity of indiscernibles:**

duo quaedam communes proprietates eorum nequaquam possit
(two distinct things cannot have all their properties in common).

Given $\Gamma \vdash A : \text{Set}$ in $\lambda C$, we can define

$$\text{Eq}_A \triangleq \lambda x, y : A \ (\Pi P : A \to \text{Prop} \ (Px \leftrightarrow Py))$$

satisfying $\Gamma \vdash \text{Eq}_A : A \to A \to \text{Prop}$ and giving a well-behaved (but not extensional) equality predicate for elements of type $A$. 
Leibniz equality in $\lambda C$

Gottfried Wilhelm Leibniz (1646–1716),

**identity of indiscernibles**: 

*duo quaedam communes proprietates eorum nequaquam possit* (two distinct things cannot have all their properties in common).

Given $\Gamma \vdash A : \text{Set}$ in $\lambda C$, we can define

$$\text{Eq}_A \triangleq \lambda x, y : A (\Pi P : A \to \text{Prop} (P x \leftrightarrow P y))$$

satisfying $\Gamma \vdash \text{Eq}_A : A \to A \to \text{Prop}$ and giving a well-behaved (but not extensional) equality predicate for elements of type $A$.

$$p \leftrightarrow q \triangleq (p \Rightarrow q) \land (q \Rightarrow p)$$
Extensionality

Functional extensionality:

\[ \text{FunExt}_{A,B} \triangleq \Pi f,g : A \to B ( \Pi x : A (\text{Eq}_B (f x) (g x))) \to \text{Eq}_{A \to B} f g) \]
Functional extensionality:

\[
\text{Ext}_{A,B} \triangleq \Pi f, g : A \to B \left( (\Pi x : A (\text{Eq}_B (f x) (g x))) \to \text{Eq}_{A \to B} f g \right)
\]

If \( \Gamma \vdash A, B : \text{Set} \) in \( \lambda C \), then \( \Gamma \vdash \text{Ext}_{A,B} : \text{Prop} \) is derivable, but for some \( A, B \) there does not exist a pseudo-term \( t \) for which \( \Gamma \vdash t : \text{Ext}_{A,B} \) is derivable.
Extensionality

**Functional extensionality:**

\[\text{FunExt}_{A,B} \triangleq \Pi f, g : A \to B \left( (\Pi x : A (\text{Eq}_B (f x) (g x))) \to \text{Eq}_{A \to B} f g \right)\]

If \( \Gamma \vdash A, B : \text{Set} \) in \( \lambda C \), then \( \Gamma \vdash \text{Ext}_{A,B} : \text{Prop} \) is derivable, but for some \( A, B \) there does not exist a pseudo-term \( t \) for which \( \Gamma \vdash t : \text{Ext}_{A,B} \) is derivable.

**Propositional extensionality:**

\[\text{PropExt} \triangleq \Pi p, q : \text{Prop} \left( (p \leftrightarrow q) \to \text{Eq}_{\text{Prop}} p q \right)\]

\( \Diamond \vdash \text{PropExt} : \text{Prop} \) is derivable in \( \lambda C \), but there does not exist a pseudo-term \( t \) for which \( \Diamond \vdash t : \text{PropExt} \) is derivable.
Extensionality

This is a weak form of Voevodsky's Univalence Axiom - currently a HOT topic in type theory research (Homotopy Type Theory)

Propositional extensionality:

\[ \text{PropExt} \triangleq \Pi p, q : \text{Prop} \ ((p \leftrightarrow q) \rightarrow \text{Eq}_\text{Prop} p q) \]

\[ \boxdot \vdash \text{PropExt} : \text{Prop} \] is derivable in \( \lambda C \), but there does not exist a pseudo-term \( t \) for which \( \boxdot \vdash t : \text{PropExt} \) is derivable.