Lectures on Dependent Type Theories

Peter Aczel
Manchester University

Dresden
June 30, July 1, 2003
Lecture One

Simple Type Theory

(L \rightarrow)

(\beta \rightarrow) Church

(\gamma \rightarrow) Curry

(\zeta \rightarrow)

Some meta-theory for (\beta \rightarrow)

Inversion & Context
Substitution
Computation
• In **classical** mathematics a proposition is something that is true or false.

• In **constructive** mathematics a prop. is something which may have proofs and is true if it has a proof.

  - A proof of $A \rightarrow B$ is a proof of $B$, assuming given a proof of $A$.

The proof-object thesis

Proofs can be adequately represented by suitable mathematical objects called proof-objects. So a prop. can be represented by the type of its proof objects.

$A \rightarrow B$ is the type of functions from $A$ to $B$; i.e. things that determine a proof-object of $B$ given a proof object of $A$.  


Intuitionistic Implicational Logic: ($\rightarrow$)

Atomic Formulae, $A_0$

Formulae, $A : = A_0 \mid (A \rightarrow A)$

Formula Contents, $\Sigma = A_1, \ldots, A_n$ ($n \geq 0$)

Form of Judgment, $\Sigma \vdash A$

**Rules of Inference**

\[
\frac{\Sigma \vdash A \quad (A \text{ in } \Sigma)}{\Sigma, A \vdash B}
\]

\[
\frac{\Sigma, A \vdash B}{\Sigma \vdash (A \rightarrow B)}
\]

\[
\frac{\Sigma \vdash (A \rightarrow B) \quad \Sigma \vdash A}{\Sigma \vdash B}
\]

→ associates to the right

E.g. $A_1 \rightarrow A_2 \rightarrow A_3$ abbreviates

$$(A_1 \rightarrow (A_2 \rightarrow A_3))$$
Examples

\[(K) \quad \vdash A \rightarrow B \rightarrow A \quad \text{(in \,(L\rightarrow))}\]

\[
\frac{A, B \vdash A}{A \vdash B \rightarrow A}
\]

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

\[(S) \quad \vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C\]

Let $\Gamma = (A \rightarrow B \rightarrow C), (A \rightarrow B), A$

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

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

\[
\frac{\Gamma \vdash C}{(A \rightarrow B \rightarrow C), (A \rightarrow B) \vdash A \rightarrow C}
\]

\[
\frac{(A \rightarrow B \rightarrow C) \vdash (A \rightarrow B) \rightarrow A \rightarrow C}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C}
\]
Pierce's Law

\[(p \rightarrow q) \rightarrow p \rightarrow p\]

is a classical tautology that cannot be proved.

An attempt

\[
\frac{(p \rightarrow q) \rightarrow p, p \vdash q}{(p \rightarrow q) \rightarrow p \vdash p \rightarrow q}
\]

\[
(p \rightarrow q) \rightarrow p \vdash p \\
\vdash ((p \rightarrow q) \rightarrow p) \rightarrow p
\]

Adding as axioms

\[\Gamma \vdash ((A \rightarrow B) \rightarrow A) \rightarrow A\]

does give a complete axiomatisation of the classical implicational tautologies.
Simple Type Theory: \((\alpha \rightarrow \alpha)_\text{church}\)
- à la Church

- Variables, \(x\)
- Terms of the **untyped** \(\lambda\)-calculus

\[
M ::= x \mid \lambda x. M \mid (MM)
\]

- Application associates to the left
e.g. \(M_0 M_1 M_2\) abbreviates

\(((M_0 M_1) M_2)\)

- \(\lambda xyz. M\) abbreviates

\(\lambda x. \lambda y. \lambda z. M\)

etc ...

- Atomic types, \(A_0\)
  Types, \(A ::= A_0 \mid (A \rightarrow A)\)

- Each variable \(x\) is given a type \(\text{ty}(x)\)
  so that there are infinitely many variables of each type.
The Church continues.

Form of judgment, $M : A$

**Term Formation Rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Type</th>
<th>(\lambda x.M : A \to B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x : A)</td>
<td>(\text{ty}(x) = A)</td>
<td></td>
</tr>
<tr>
<td>(M : B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\lambda x.M : A \to B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M : A \to B), (N : A)</td>
<td></td>
<td>((MN) : B)</td>
</tr>
</tbody>
</table>

**Uniqueness of Types**

If $M : A_1$, and $M : A_2$ in \((\lambda \to)\) Church, then $A_1 = A_2$.

\((\lambda \to)\) Church is **monomorphic**.
Formulae-as-Types

Identify the formulae of \((L \to)\) with the types of \((\lambda \to)\)

Theorem

Let

\[ x_1, \ldots, x_n \text{ be distinct variables} \]
\[ A_i = \text{ty}(x_i) \quad (i = 1, \ldots, n) \]
\[ \Sigma = A_1, \ldots, A_n \]

Then

\[ \Sigma \vdash A \text{ in } (L \to) \text{ iff} \]
\[ M : A \text{ in } (\lambda \to)_{\text{Church}} \text{ for some term } M \text{ such that } \var(M) \subseteq \{x_1, \ldots, x_n\} \]

\[ \var(M) = \text{set of variables that occur free in } M \]

This is the Curry-(deBruijn-Howard)
correspondence

\[ (L \to) \sim (\lambda \to) \]
Simple Type Theory: \((\lambda \to)\) Curry
- à la Curry

Variable declaration contexts
\[
\Gamma = x_1 : A_1, \ldots, x_n : A_n \quad (n \geq 0)
\]
with \(x_1, \ldots, x_n\) distinct.

\[
\text{forget } x \mapsto \text{ty}(x)
\]

Term Formation Rules

\[
\begin{align*}
\Gamma \vdash x : A & \quad (x : A \text{ in } \Gamma) \\
\Gamma, x : A \vdash M : B \\
\hline
\Gamma \vdash \lambda x. M : A \to B \\
\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A \\
\hline
\Gamma \vdash (M N) : B
\end{align*}
\]

\((\lambda \to)\) Curry is \underline{polymorphic}.

e.g. \(\vdash \lambda x. x : A \to A\)
for any type \(A\).
\((\to)_{\text{Curry}} \sim (\to)_{\text{Church}}\)

**Theorem**

Let \(x_1, \ldots, x_n\) be distinct, \((n \geq 0)\),
\[A_i = ty(x_i) \quad (i=1, \ldots, n),\]
\[\Gamma = x_1 : A_1, \ldots, x_n : A_n.\]

Then \(\Gamma \vdash M : A\) in \((\to)_{\text{Curry}}\) iff

\[\{\text{var}(M) \subseteq \{x_1, \ldots, x_n\}\} \text{ and } M : A\text{ in } (\to)_{\text{Church}}\].

**Corollary**

Let \(\Sigma = A_1, \ldots, A_n\).
Then \(\Sigma \vdash A\) in \((\to)\) iff

\[\{\Gamma \vdash M : A\text{ in } (\to)_{\text{Curry}} \text{ for some } M\}\]

e.g. let \(K = \lambda xy. x\)
\(S = \lambda xyz. xz(yz)\)

Then, in \((\to)_{\text{Curry}}\),

\[\vdash K : A \to B \to A\]
\[\vdash S : (A \to B \to C) \to (A \to B) \to A \to C\]
The Simple Type Theory \( (\rightarrow) \)

forget. \( \times \xrightarrow{ty} (\times) \)

Pre-terms, \( M ::= x | \lambda x : A . M | (M M) \)

Term Formation Rules

<table>
<thead>
<tr>
<th>( \Gamma \vdash x : A ) (( x : A ) in ( \Gamma ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, x : A \vdash M : B )</td>
</tr>
<tr>
<td>( \Gamma \vdash \lambda x : A . M : A \rightarrow B )</td>
</tr>
<tr>
<td>( \Gamma \vdash M : A \rightarrow B ) ( \Gamma \vdash N : A )</td>
</tr>
<tr>
<td>( \Gamma \vdash (M N) : B )</td>
</tr>
</tbody>
</table>

Theorem \( \Gamma \vdash M : A \) in \( (\rightarrow) \)\text{Curry} iff \( \Gamma \vdash N : A \) in \( (\rightarrow) \)

for some \( N \) such that \( M = N^- \)

\( N^- \) is \( N \), with every \( : A \) removed

e.g. if \( N = \lambda x : (A \rightarrow B \rightarrow C) . \lambda y : (A \rightarrow B) \)

\( \cdot \lambda z : A . \ x z (y z) \)

then \( N^- = \lambda x y z . x z (y z) \)
Inversion Properties

- $\Gamma \vdash x : A \quad \Rightarrow \quad x : A$ in $\Gamma$

- $\Gamma \vdash \lambda x : A \cdot M : C \quad \Rightarrow \quad [\Gamma, x : A \vdash M : B$ and $C = (A \rightarrow B)$ for some $B$]

- $\Gamma \vdash (MN) : B \quad \Rightarrow \quad [\Gamma \vdash M : A \rightarrow B$ and $\Gamma \vdash N : A$ for some $A$]

- If $\Gamma \vdash M : A$ in $(\Rightarrow)$ then the type $A$ and the derivation tree of $\Gamma \vdash M : A$ in $(\Rightarrow)$ are uniquely determined by $\Gamma$ and $M$. 
Formulae-as-Types again

\[ \Sigma = A_1, \ldots, A_n \]
\[ \Gamma = x_1 : A_1, \ldots, x_n : A_n \]
\[ \Sigma \vdash A \text{ in } (L \Rightarrow) \text{ iff } [\Pi \vdash M : A \text{ in } (I \Rightarrow) \text{ for some } M] \]

- \( M \) is a **proof object** representing a derivation tree of \( \Sigma \vdash A \) in \( (L \Rightarrow) \).

- \( M \) is an **algorithm/program** for computing a value of type \( A \) given values of types \( A_1, \ldots, A_n \).
Decision Problems for (→)

\[ \Gamma \vdash M : A \]  type checking
\[ \Gamma \vdash M : ? \]  type synthesis
\[ \Gamma \vdash ? : A \]  type inhabitation

All decidable!

But type inhabitation becomes undecidable for most extensions of (→)
Context Properties

\[ \text{var}(M) = \text{set of variables that occur free in } M \]

\[ \text{var}(x_1:A_1, \ldots, x_n:A_n) = \{ x_1, \ldots, x_n \} \]

\[ \Gamma_1 \subseteq \Gamma_2 \text{ if } \{ x:A \text{ in } \Gamma_1 \Rightarrow x:A \text{ in } \Gamma_2 \} \]

Free Variables

\[ \Gamma \vdash M:A \Rightarrow \text{var}(M) \subseteq \text{var}(\Gamma) \]

Weakening  If \( \Gamma \subseteq \Gamma' \) then

\[ \Gamma \vdash M:A \Rightarrow \Gamma' \vdash M:A \]

Strengthening  If \( \Gamma \subseteq \Gamma' \) and \( \text{var}(M) \subseteq \text{var}(\Gamma) \)

\[ \Gamma' \vdash M:A \Rightarrow \Gamma \vdash M:A \]
Substitution Properties

- Identify terms up to relabelling of bound variables.
- \( M[N_1, \ldots, N_n/x_1, \ldots, x_n] \) is the result of simultaneously substituting \( N_i \) for free occurrences of \( x_i \) in \( M \) for \( i=1, \ldots, n \), relabelling bound variables so as to avoid variable clashes. (\( x_1, \ldots, x_n \) distinct)

Substitution Theorem for \((\exists \rightarrow)\)

Let \( \Gamma = x_1: A_1, \ldots, x_n: A_n \)
\( \Xi = x_1, \ldots, x_n \)
\( \eta = N_1, \ldots, N_n \)

If \( \Gamma \vdash M : B \) then
\( \Gamma' \vdash N_j : A_j \) (\( j=1, \ldots, n \))
\[ \Gamma' \vdash M[N_j/\Xi] : B \]
Cut  If \( \Gamma, x : A \vdash M : B \) then
\[ \Gamma \vdash N : A \implies \Gamma \vdash M[N/x] : B \]

**Subject Contraction**

\( \beta \)-contraction
\[ \Gamma \vdash (\lambda x : A. M \ N) : B \]
\[ \implies \Gamma \vdash M[N/x] : B \]

\( \eta \)-contraction  If \( x \in \text{var}(M) \)
\[ \Gamma \vdash \lambda x : A. (Mx) : A \to B \]
\[ \implies \Gamma \vdash M : A \to B \]

\((\beta)\)  \( (\lambda x : A. M \ N) \longrightarrow M[N/x] \)

\((\eta)\)  \( \lambda x : A. (Mx) \longrightarrow M \ (x \notin \text{var}(M)) \)

- \( M \to_\beta M' \) if \( M' \) is obtained from \( M \) by contracting a single subterm of \( M \) that is a \( \beta \)-redex.  \( \to_\eta \) similarly

- \( \to_\beta = \to_\beta \cup \to_\eta \)

- \( \to_\beta = (\to_\beta)^* \)
Computation Properties

Subject Reduction If \( M \xrightarrow{\beta_n} M' \) then
\[ \Gamma \vdash M : A \implies \Gamma \vdash M' : A \]

Church Rosser
If \( M \xrightarrow{\beta_n} M_1 \) and \( M \xrightarrow{\beta_n} M_2 \) then
\[ M_1 \xrightarrow{\beta_n} M' \text{ and } M_2 \xrightarrow{\beta_n} M' \text{ for some } M'. \]

Strong Normalisation If \( \Gamma \vdash M : A \)
then \( M \) is strongly normalising; i.e.
there is no infinite computation
\[ M \xrightarrow{\beta_n} M' \xrightarrow{\beta_n} M'' \xrightarrow{\beta_n} \ldots \]

Normalisation If \( \Gamma \vdash M : A \)
then \( M \) normalises; i.e.
\[ M \xrightarrow{\beta_n} \bar{M} \]
for some \( \bar{M} \) that is normal; i.e.
has no redex subterm
Some non-dependent extensions of \((\lambda \to)\)

\[ L(\to, \& , \lor , \perp , T) \]

\[ \lambda(\to, x, +, N_0, N_1, \ldots) \]

Primitive Recursion

\[ \lambda(\to, \ldots, N) \]

Introducing Dependent Types

\[ R_k(M, A_1, \ldots, A_n) \]

The type theory \( ML^- \)

\( ML^- \) and Gödel's \( T \)

Adding \( T \) and \( \Sigma \) types
In Constructive Maths:

To prove \( A \rightarrow B \),
prove \( B \), assuming \( A \)

To prove \( A \land B \),
prove \( A \) and prove \( B \)

To prove \( A \lor B \)
choose one of them & prove it

To prove \( \bot \)
'do the impossible'

To prove \( T \)
'done'

With the proof-objects thesis

\[ A \land B = A \times B \text{ cartesian product} \]

\[ A \lor B = A + B \text{ disjoint union} \]

\[ \bot = \emptyset \text{ empty type} \]

\[ T = \mathbb{N}, \text{ unit type} \]
Intuitionistic Propositional Logic:

\[ \mathcal{L} (\to, \land, \lor, \bot, \top) \]

\[ A :: = A_0 \mid (A \to A) \mid (A \land A) \mid (A \lor A) \mid \bot \mid \top \]

Contents, \( \Sigma = A_1, \ldots, A_n \)

Form of Judgment, \( \Sigma \vdash A \)

Rules of Inference

\[
\frac{\Sigma, A \vdash B}{\Sigma \vdash A \to B} \quad \frac{\Sigma \vdash A \to B \quad \Sigma \vdash A}{\Sigma \vdash B}
\]

\[
\frac{\Sigma \vdash A_i \quad (i = 1, 2)}{\Sigma \vdash A_1 \land A_2} \quad \frac{\Sigma \vdash A_1 \land A_2}{\Sigma \vdash A_i \quad (i = 1, 2)}
\]

\[
\frac{\Sigma \vdash A_1 \lor A_2}{\Sigma \vdash A_1 \lor A_2} \quad \frac{\Sigma \vdash A_1 \vdash C \quad \Sigma \vdash A_2 \vdash C}{\Sigma \vdash C}
\]

\[
\Sigma \vdash \top
\]

\[
\neg A = (A \to \bot)
\]

\[
A \leftrightarrow B = (A \to B) \land (B \to A)
\]
\( \lambda (\to, \times, +, N_0, N_1, \ldots) \)

Types, \( A ::= \ldots | (A \times A) | (A + A) | N | N_1 | \ldots \)

- \( A_1 \times A_2 \) is a cartesian product type
- \( A_1 + A_2 \) is a disjoint union type
- \( N_k \) is a \( k \)-element type (\( k = 0, 1, \ldots \))

Formulae-as-types

- \( A_1 \land A_2 = A_1 \times A_2 \)
- \( A_1 \lor A_2 = A_1 + A_2 \)
- \( \top = \mathbb{N}_1 \)
- \( \bot = N_0 \)

Preterms

\( M ::= \ldots | \pi_i(M, M) | \pi_i(M) \quad (i = 1, 2) \)

- \( \text{in}_i(M) \)
- \( \text{cases}(M, (x)M, (x)M) \)
- \( \text{in}_k(M) \)
- \( R_k(M, M, \ldots, M) \quad (k = 0, 1, \ldots, i = 1, \ldots, k) \)
In order to keep uniqueness of types
and $\text{R}^n(M)$, $\text{R}^n_{\text{C}}(M)$, 
and in $(M)$ should perhaps be in $\text{R}^n_{\text{C}}(M)$
so in $(M)$ has been omitted.

Some type info has been omitted.

\[
\frac{\text{R}^n(M, M', \ldots, M'_n) : C}{\vdash M : N^v, L \vdash M : C (\leq 1', 2')}
\]

\[
\frac{\text{cases} (M, (x)M', M'_2) : C}{\vdash M : A + A_2, L \vdash M : C (\leq 1', 2')}
\]

\[
\frac{\text{ex} (M) : A}{\vdash M : A, (\leq 1', 2')}
\]

\[
\frac{M : A, (\leq 1', 2')}{\vdash \text{new} M(A, M') : A, (\leq 1', 2')}
\]

New Rules
Contractions

\((\lambda x : A. M \ N) \rightsquigarrow M[N/x]\)

\(\pi_i(\Pi(M_1, M_2)) \rightsquigarrow M_i \ (i = 1, 2)\)

\(\text{cases}(\text{in}_i(M), (x_i)M_i, (x_i)M_i) \rightsquigarrow M_i[M/x_i] \ (i = 1, 2)\)

\(\text{R}_i(M_1, M_2, \ldots, M_k) \rightsquigarrow M_i \ (i = 1, \ldots, k)\)

No \(\eta\)-contractions are used!
Primitive Recursion

$M ::= x | 0 | s(M) | R(M, M, (x, x)M)$

Contractions

$R(0, M_0, (x, y)M_i) \rightsquigarrow M_0.$

$R(s(M), M_0, (x, y)M_i)$

$\rightsquigarrow M_i[M, R(M, M_0, (x, y)M_i)/x, y]$  

$n = \overbrace{s(\ldots s(s(0))\ldots)}^{n} \quad (n = 0, 1, \ldots)$

Theorem \(f : \text{nat}^k \to \text{nat}\) is primitive recursive iff there is a term \(M\) with \(\text{var}(M) \subseteq \{x_1, \ldots, x_k\}\) such that \(f(n_1, \ldots, n_k) = n\)

\(\iff M[n_1, \ldots, n_k/x_1, \ldots, x_k] \to n\)

Note: Church-Rosser and Strong Normalisation hold.

Note: The theorem could be used as a definition.
Adding the type \( N \)

\[
A ::= \ldots \mid N \\
M ::= \ldots \mid O \mid s(M) \mid R(M, M, (x, x)M)
\]

Contractions for \( R \) as before

**New Rules**

\[
\Gamma \vdash O : N \\
\frac{\Gamma \vdash M : N}{\Gamma \vdash s(M) : N}
\]

\[
\Gamma \vdash M : N \quad \Gamma \vdash M_0 : C \quad \Gamma, x : N, y : C \vdash M : C \\
\frac{\Gamma \vdash R(M, M_0, (x, y)M) : C}{\Gamma \vdash R(M, M_0, (x, x)M) : C}
\]
Introducing Dependent Types

- In predicate logic formulae generally depend on certain variables—those that occur free in it.
- With formulae-as-types we want to allow types to depend on variables ranging over other types.
- To introduce such dependency we use type formers $\text{IR}_k$ ($k=0,1,\ldots$)

Intuitive idea

Given types $A_1,\ldots, A_k$ we want a type $F(x)$ depending on $x : \text{IR}_k$ such that $F(i,k) = A_i$ ($i=1,\ldots,k$)

- This $F(x)$ will be $\text{IR}_k(\forall x A_1,\ldots, A_k)$ with contractions $\text{IR}_k(i,k,A_1,\ldots, A_k) \rightarrow A_i$ ($i=1,\ldots,k$)
- In general $\text{IR}_k(M,A_1,\ldots, A_k)$ will not be well-formed if $M$ is not
- Also it can depend on variables of any type.
• We need a notion of pre-type.
• Pre-types and preterms have to be defined simultaneously:
  \[ A :: = \ldots | R_k(M, A, \ldots, A) | \ldots \]
  \[ M :: = \ldots | \lambda x: A. M | \ldots \]

• Forms of Judgment
  \[ \Gamma \vdash \cdot \]
  \[ \Gamma \vdash A \text{ type} \]
  \[ \Gamma \vdash M : A \]

General Rules of Inference

Contexts
\[ \Gamma \vdash \cdot \]

\[ \frac{\Gamma \vdash A \text{ type} \quad (x \in \text{var}(A))}{\Gamma, x : A \vdash \cdot} \]

Variables
\[ \frac{\Gamma \vdash \cdot}{\Gamma \vdash x : A} \]

Conversion
\[ \frac{\Gamma \vdash M : A \quad \Gamma \vdash B \text{ type} \quad (A \text{ conv } B)}{\Gamma \vdash M : B} \]

\text{conv} is the equivalence relation generated by the one-step reduction relation \( \Rightarrow \).
The Type Formation Rules

\[ \Gamma \vdash A_i \text{ type} \quad (i=1,2) \quad (\square \in \{\to, \times, +\}) \]

\[ \frac{}{\Gamma \vdash (A_1 \square A_2) \text{ type}} \]

\[ \frac{\Gamma \vdash \cdot}{\Gamma \vdash N_k \text{ type}} \quad (k = 0, 1, \ldots) \]

\[ \frac{\Gamma \vdash \cdot}{\Gamma \vdash N \text{ type}} \]

\[ \frac{\Gamma \vdash M : N_k \quad \Gamma \vdash A_i \text{ type} \quad (i=1,\ldots,k)}{\Gamma \vdash TR_k(M, A_1,\ldots,A_k) \text{ type}} \]

\[ (k = 0, 1, \ldots) \]
Term Formation Rules

- For $\rightarrow$, $x : \text{as before}$

- For $+$:
  \[
  \frac{\Gamma \vdash M : A_1, \Gamma \vdash A_2 \text{type}}{\Gamma \vdash \text{in}_1(M) : A_1 + A_2} \quad \frac{\Gamma \vdash A_1 \text{type}}{\Gamma \vdash \text{in}_2(M) : A_1 + A_2}
  \]

  \[
  \Gamma, z : A_1 + A_2 \vdash C \text{type} \\
  \Gamma \vdash M : A_1 + A_2 \\
  \Gamma, x : A_i \vdash M_i : C[\text{in}_i(x_i)/z] \ (i = 1, 2)
  \]

  \[
  \Gamma \vdash \text{cases}^*(M, (x_1)M_1, (x_2)M_2) : C[M/z]
  \]

- For the $N_k$ and $N$: the intro rules as before

- For the elim rules:
  \[
  \Gamma, z : N_k \vdash C \text{type} \\
  \Gamma \vdash M : N_k \\
  \Gamma \vdash M_i : C[ik/z] \ (i = 1, \ldots, k)
  \]

  \[
  \Gamma \vdash R_k^*(M, M_1, \ldots, M_k) : C[M/z]
  \]

- For $R^*$:
  \[
  \Gamma, z : N \vdash C \text{type} \\
  \Gamma \vdash M : N \\
  \Gamma \vdash M_0 : C[0/z] \\
  \Gamma, x : N, y : C[x/z] \vdash M_1 : C[\text{x}(x)/z]
  \]

  \[
  \Gamma \vdash R^*(M, M_0, (x,y)M_1) : C[M/z]
  \]
• The elimination rule for \( \mathbb{N} \) is the type theoretic version of the following mathematical induction rule.

\[
\Sigma \vdash C[0/z] \quad \Sigma, C[z/z] \vdash C[s(z)/z]
\]

\[\Sigma \vdash C[M/z]\]

• Gödel's Dialectica interpretation gives a reduction of HA to his system T of primitive recursive functions of finite type.

Theorem There is a reduction of T to the type theory \( \lambda(\to, x, +, \mathbb{N}, \mathbb{N}, \ldots, \mathbb{N}, \mathbb{R}, \mathbb{R}, \ldots) \).

Call this dependent type theory ML-
I ended lecture 2 with the claim that Gödel's T can be reduced to
the dependent type theory

\[ ML^- = \lambda (\rightarrow, x, +, N_0, N_1, \ldots, N, R_0, R_1, \ldots) \]

This is really a conjecture whose details need checking!

The idea

1) The finite types of T can be represented as the types of ML^- built up from
   \( N \) using \( \rightarrow \).

2) The terms of T can be represented as terms of ML^- using variables of
   finite type, application, abstraction and \( R \).

3) The formulae of T can be taken to be in the form \( (t = o) \) where \( t \)
   is a term of T of type \( N \).

So we need to represent

\( (x = o) \) as a dependent type \( \exists(x) \)
depending on \( x : N \) which has
suitable properties.

Let \( \exists(x) = \Pi R_2 (R(x, 1_2, (u, v) 2_2), N_1, N_0) \)

So \( \exists(0) \rightarrow_\beta N_1 \) and \( \exists(s(M)) \rightarrow_\beta N_0 \)
Lecture 3

$\Pi$, $\Sigma$ types

Intuitionistic type theory

Type Universes

Inductive types

The type of iterative sets
The quantifiers

In constructive mathematics

1. to prove \( \forall x : A \ B(x) \)
   
   prove \( B(x) \) for arbitrary \( x : A \)

2. to prove \( \exists x : A \ B(x) \)
   
   choose \( a : A \) and prove \( B(a) \)

With the proof-objects thesis

\( \forall x : A \ B(x) = \text{type of all functions } f \)
   
   defined on \( A \) such that
   
   \( f(x) : B(x) \) for \( x : A \)

\( = (\Pi x : A) B(x) \)

\( \exists x : A \ B(x) = \text{type of all pairs } (a, b) \)
   
   such that \( a : A, b : B(a) \)

\( = (\Sigma x : A) B(x) \)
\( \Pi \) and \( \Sigma \) types

\[ A ::= \ldots | (\Pi x:A)A | (\Sigma x:A)A \]

\( \langle (\Pi x:A)B(x) \rangle \) is the \{cartesian product\}
\( \langle (\Sigma x:A)B(x) \rangle \) is the \{disjoint union\}

of the family of types \( B(x) \) for \( x:A \)

**Type Formation**

\[ \Gamma, x:A \vdash B \text{ type} \]

\[ \frac{}{\Gamma \vdash (Q x:A)B \text{ type}} \quad (Q \in \{\Pi, \Sigma\}) \]

**Term Formation**

**Intro Rules**

\[ \Gamma, x:A \vdash M : B \]

\[ \frac{}{\Gamma, \lambda x:A. M : (\Pi x:A)B} \]

\[ \Gamma, x:A \vdash B \text{ type} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : B[\lambda x:A.M] \]

\[ \frac{}{\Gamma \vdash \pi^i(M, N) : (\Sigma x:A)B} \]
Elim Rules

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

\[ \Gamma, \varepsilon : (\Sigma x : A)B \vdash C \text{ type} \]
\[ \Gamma \vdash M : (\Sigma x : A)B \]
\[ \Gamma, x : A, y : B \vdash N : C[\pi(x,y)/\varepsilon] \]
\[ \Gamma \vdash \text{split} (M, (x,y)N) : C[M/\varepsilon] \]

\[ M ::= \cdots \mid \text{split} (M, (x,y)N) \]
\[ \text{split} (\pi(M_0, M_1), (x,y)N) \]
\[ \leadsto N[M_0, M_1/x, y] \]

- With formulae-as-types
  \[ (\forall x : A)B = (\Pi x : A)B \]
  \[ (\exists x : A)B = (\Sigma x : A)B \]

Note: \( A \to B / A \times B \) are the special cases of \((\Pi x : A)B / (\Sigma x : A)B\) when \(x \notin \text{var}(B)\). \( \pi_1, \pi_2 \) can be defined. So the rules for \( \to, \times \) are redundant.
• ML is the dependent type theory extending ML$^-$ with rules for the IT and Σ types.

• Although $HA \preceq T \preceq ML \preceq ML$ there is a direct translation $HA \preceq ML$

  using formulae-as-types and so avoiding the complexities of Gödel’s coding.

• But instead ML has a more complicated type theory — being dependent.
  Gödel’s $T$ just uses finite simple types built from $N$.

• Martin-Löf has developed a ‘meaning explanation’ intended to justify the rules of his type theories.

• This corresponds to Gödel’s Dialectica discussion about the computable functions of finite type.

• Note: $ML \preceq HA$ e.g. via realizability.
• ML is a variant of the core basic type theory of Per Martin-Löf.

• His **intuitionistic type theory** does not use **my** $\text{Re}$ or the conversion rule, but instead has

  - (intensional) equality types
    \[ I(A, M_1, M_2) \]
  - judgmental equality
    \[ \Gamma \vdash A_1 = A_2 \]
    \[ \Gamma \vdash M_1 = M_2 : A \]

• I think that the equality types are unnecessary for most purposes and I prefer to avoid them

• Judgmental equality seems important for some purposes; e.g. Martin-Löf's meaning explanations. But it requires a large number of additional rules. Many type theories do not use them, but use Conversion; e.g. PTSs and the type theories implemented in Coq and Lego.
- Martin-Löf has extended the core type theory with
  - predicative type universes
  - inductive types
- These greatly increase the proof theoretic strength of the type theory while keeping it (generalised) predicative.
Type Universes

A type universe \( U \) is a type whose objects are
\[ \{ \text{types} \} \quad \text{à la Russell} \]
\[ \{ \text{codes of types} \} \quad \text{à la Tarski} \]

The general rules are
\[ \Gamma \vdash \cdot \quad \Gamma \vdash U \text{ type} \]
and
\[ \Gamma \vdash M : U \quad \text{à la Russell} \]
\[ \Gamma \vdash M : U \quad \text{à la Tarski} \]
\[ \Gamma \vdash \Pi (M) \text{ type} \]

The Tarski version keeps a distinction between terms and types.

A universal type universe

Add the rule
\[ \Gamma \vdash A : \text{type} \quad \text{only a Russell version} \]
\[ \Gamma \vdash A : U \]

We can derive \( \Gamma \vdash U : U \).

The theory is inconsistent!

Girard's Paradox (a version of Burali-Forti paradox)
An impredicative type universe

Add the rule

\[
\Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash M : U
\]

\[
\Gamma \vdash (\prod x : A)M : U
\]

This is impredicative because \( A \) can be \( U \) itself, so that an element of \( U \) is formed by quantifying over \( U \).

\[
\Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash M : U
\]

\[
\Gamma \vdash (\forall x : A)M : U
\]

with contraction

\[
\Pi((\forall x : A)M) \leftrightarrow (\forall x : A)\Pi(M)
\]

• Both forms can be consistently added to ML. But adding similar rules for \( \Sigma \) types again lead to inconsistency.
A predicative type universe

The rules for a predicative type universe arise by a process of 'reflection' on the rules for forming types of a given type theory.

We illustrate with the formation rules for $\Pi$, $+$, $\Pi$.

**Russell version**

\[
\frac{\Gamma \vdash A : U \quad \Gamma, x : A \vdash B : U}{\Gamma \vdash (\Pi x : A)B : U}
\]

\[
\frac{\Gamma \vdash A_1 : U \quad \Gamma \vdash A_2 : U}{\Gamma \vdash A_1 + A_2 : U}
\]

**Tarski Version**

\[
\frac{\Gamma \vdash M : U \quad \Gamma, x : T(M) \vdash N : U}{\Gamma \vdash (\Pi x : T(M))N : U}
\]

\[
\frac{\Gamma \vdash M_1 : U \quad \Gamma \vdash M_2 : U}{\Gamma \vdash M_1 + M_2 : U}
\]

\[
\Pi((\Pi x : M)N) \quad \longrightarrow \quad (\Pi x : T(M))T(N)
\]

\[
T(M + M_2) \quad \longrightarrow \quad T(M) + T(M_2)
\]

\[
T(N) \quad \longrightarrow \quad N
\]
Example

$N' = (Wx : N') R_2 (x, N_0, N_1)$

is 'isomorphic' to $N$

- $O': N'$ is $\sup(\exists x, \exists x : N_0, R_2(x))$

- If $M : N'$ then $s'(M) : N'$ is
  $\sup(\exists x, \exists x : N_0, M)$
Adding \( W \)-types

These are inductive types of well-founded trees with formation and intro rules

\[
\Gamma, x : A \vdash B \text{ type} \\
\frac{}{\Gamma \vdash (Wx : A)B \text{ type}}
\]

\[
\Gamma, x : A \vdash B \text{ type} \\
\Gamma \vdash M : A \\
\Gamma \vdash N : B[M/x] \rightarrow (Wx : A)B \\
\frac{}{\Gamma \vdash \text{sup}(M,N) : (Wx : A)B}
\]

\[\cdots\]

\[\text{Sup}(M,N)\]

\[\text{(y : B[M/x])}\]

\[\text{(N y)}\]

\[\cdots\]
The type $\mathcal{V}$ of $U$-iterative sets

Given a predicative universe $U$ let

$\mathcal{V} = (Wx : U) \check{T}(x)$

It has the intro' rule

$\Gamma \vdash M : U \quad \Gamma \vdash N : T(M) \rightarrow U$

$\Gamma \vdash \text{sup}(M,N) : \mathcal{V}$

- $\mathcal{V}$ can be used as a 'universe of iterative sets' for an axiomatic set theory:
  - $\text{sup}(M,N)$ is the set whose elements are sets $(Nx)$ for $x : T(M)$.
  - e.g. the empty set $\emptyset$ is
    $\text{sup}(N_0, \forall x : N_0. R_0(x)) : \mathcal{V}$
  - the unordered pair $\{M_1, M_2\}$ is
    $\text{sup}(N_2, \forall x : N_2. R_2(x, M_1, M_2)) : \mathcal{V}$

etc...
To interpret the extensibility axiom \( \forall x (x \in x \leftrightarrow x \in y) \rightarrow (x = y) \), we need \((x = y)^V : V\) for \(x, y : V\).

This can be defined using the elimination rule for \(V\)-types so that
\[
[sup(M, N) = sup(M_2, N_2)]
\]
\[
= (\forall x : M)(\exists x : M_2)[(N_2 x_e) = (N_2 x_t)]
\]
\[
\land (\forall x : M_2)(\exists x : M_1)[---]
\]

To interpret \(\epsilon\) we use \((x \in (y)) : V\) for \(x, y : V\) defined so that
\[
[x \in \sup(M, N)] = (\exists y : M)[x = \epsilon_N y]
\]


\(V\), with \(- = \epsilon\), \(\epsilon\) interprets CZF.

CZF is an axiom system for constructive set theory.

See notes of Aczel & Rathjen

http://www.math.kva.se
Pure Type Systems

Preterms
Assume a set of sorts, $s$

\[ M ::= x \mid s \mid \lambda x : M. M \mid (M M) \mid (\Pi x : M) M \]

Contractions

\[ (\lambda x : A. M \ N) \rightsquigarrow M[\ N/x\ ] \]

General Rules

(Conversions)

\[ \Gamma \vdash A : \text{Type} \]

(Variables)

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

(Abstraction)

\[ \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A. b : (\Pi x : A) B} \]

(Applications)

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

(Types)

\[ \frac{}{\Gamma \vdash A : s} \quad A \text{ type} \]
Special Rules

Assume given

Axioms: Set $A$ of pairs of sorts

Rules: Set $R$ of triples of sorts

\[
\text{(Axion)} \quad \frac{\Gamma \vdash \cdot}{\Gamma \vdash \pi_i : s_i}
\]

\[
\text{(Product)} \quad \frac{\Gamma \vdash A : s, \quad \pi_i : A \vdash B : s_i}{\Gamma \vdash (\pi_i : A)B : s_i} \quad \iff \quad ((s, s_i, s) \in R)
\]

A triple $(S, A, R)$ is a PTS specification, where $S$ is the set of sorts

write axioms $(s_1, s_2) \in A$ as $s_1 : s_2$

write rules $(s_1, s_2, s_3)$ as $(s_1, s_2)$
Barendregt's Cube

A fine-grained analysis of the calculus of constructions.

\[ S = \{ *, \square \} \quad A = \xi (*, \square) \]

- \( \lambda \rightarrow \) is simple type theory
  - rule \((*, *)\)
- \( \lambda 2 \) is Girard's system \( F \)
  - rules \((*, *), (\square, *)\)
  - As a logic this is second order intuitionistic implicational logic
- \( \lambda \omega \) is Girard's system \( F^\omega \)
  - rules \((*, *), (\square, *), (0, 0)\)

These three are non-dependent.
Three Dependent Type Theories

AP
rules (*, *), (*, □)

AP2
rules (*, *), (*, □), (□, *)

AC is a version of the calculus of constructions
rules (*, *), (□, *), (*, □), (□, □)

Remaining two

AW rules (*, *), (□, □)

APw rules (*, *), (□, □), (*, □)
What are *, □ ?

In \( A \to * \) is the type of types

\[
\text{axiom } * : \square \text{ ensures that } * \text{ is a type}
\]

\( \text{rule}(\square,*) \) only gives types \( A \to A_2 \)

as no type dependency is introduced.

In \( \lambda C \) □ is the sort of types

* is an impredicative type

universe

The rule \((\square,*)\) gives the

impredicative rule, which

is also in \( \lambda 2 \) and \( \lambda \omega \).
Logical Frameworks

- These are dependent type theories in which a wide class of type theories/logics can be specified simply by listing a signature of typed symbols.

- When implemented in a proof assistant a user can then easily implement their desired type theory/logic.

- Today there are many logical frameworks:

  - PAL
  - ELF
  - LF
  - TF
  - PAL+

  From de Bruijn's Automath
  Edinburgh uses higher order syntax
  Martin-Löf
  a weak LF
  Luo (Dunham)

Robin Adams (PhD thesis-Manchester)
To appear ~ Autumn 2005?
For references
- wait for an email from me for a general list
- for specific topics send me an email - petem@cs.man.ac.uk
- look on my web page
- use google
  there is a great deal of very useful material on type theory available on the web