CS3202: Logic, Specification and Verification

CS3202-LSV 2006–07

cs3202.lec@cs.st-andrews.ac.uk

Dr. James McKinna, Rm 1.03

Dr. Stéphane Lengrand, Rm. 1.02
Lecture 4 (20/02/2007):

Intuitionistic vs. classical logic

Decorating proof-trees with variables and terms
A typo in the tutorial sheet

- Prove? ⊢ ((A ⇒ B) ⇒ B) ⇒ A

- Check whether it is a tautology. Conclusion?
A typo in the tutorial sheet

- Prove? \( \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A \)

- Check whether it is a tautology. Conclusion?
## Truth Table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \Rightarrow B$</th>
<th>$(A \Rightarrow B) \Rightarrow A$</th>
<th>$((A \Rightarrow B) \Rightarrow A) \Rightarrow A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Reasoning by contradiction

\[
\begin{align*}
\frac{[A] \quad [\neg A]}{\perp} \\
\frac{\perp}{B} \\
\frac{[(A \Rightarrow B) \Rightarrow A] \quad A \Rightarrow B}{A} \\
\frac{\perp}{\neg A} \\
\frac{\perp}{A} \\
\frac{((A \Rightarrow B) \Rightarrow A) \Rightarrow A}{A}
\end{align*}
\]

Is this correct?
Reasoning by contradiction

In fact we proved:

\[ (A \implies B) \implies A \]

\[ \neg A \]

\[ \bot \]

\[ B \]

\[ \frac{[ (A \implies B) \implies A ]}{A \implies B} \]

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

\[ \bot \]

\[ \neg \neg A \]

\[ ((A \implies B) \implies A) \implies \neg \neg A \]
Natural deduction: soundness & completeness

- Soundness: $\phi_1, \ldots, \phi_n \vdash \phi$ implies $\phi_1, \ldots, \phi_n \models \phi$.

- Completeness: $\phi_1, \ldots, \phi_n \models \phi$ implies $\phi_1, \ldots, \phi_n \vdash \phi$?

No.
Classical Logic vs. Intuitionistic Logic

- Last lectures: *Intuitionistic Logic*
  
  Semantically *incomplete* (w.r.t. truthtables)

- *Classical logic* is obtained by adding:
  
  - Reasoning by contradiction
  - Elimination of Double Negation
  - Peirce’s Law

\[
\begin{align*}
\neg A & \\
\vdots & \\
\bot & \\
\hline \ A & \\
\vdots & \\
\neg \neg A & \\
\hline \ A & \\
\hline \ A & \\
\end{align*}
\]

Semantically *complete* (w.r.t. truthtables)
Proof-terms for proof-trees
Linking open/active leaves and variables

- We use Variables to keep track of the set of open/active leaves.

- In CoQ: Variable H:A.

- Idea: Represent proofs with terms /
  Decorate inference rules & steps with terms and variable annotations

- Along proof tree: need to keep track of the link open/active leaves — variables in an Environment (a.k.a. Context):
  Environment = mapping from variables to wff (e.g. \( x : A \) for \( x \mapsto A \))

- In decorated trees: Nodes are labelled with environment+term+wff

\[ \Gamma \vdash M : A \]
Typing, proofs...

- Noooooo! same symbol as in $\phi_1, \ldots, \phi_n \vdash \phi$!

But... it will be the case (to be checked in each decorated rule later) that

There exists an $M$ & a (decorated) proof-tree of

\[ x_1 : \phi_1, \ldots, x_n : \phi_n \vdash M : \phi \]

if & only if

\[ \phi_1, \ldots, \phi_n \vdash \phi \]

- Notion of **Typing**: $\Gamma \vdash M : A$ “$M$ is of type $A$ in the environment $\Gamma$”
  = “$M$ represents a proof of $A$ under the (decorated) assumptions $\Gamma$”
Natural deduction as a Typing System

- Use of an hypothesis:

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

On the l.-h. side, the \textit{wff} $A$ is declared to be available as an open assumption, decorated with $x$.

The \textit{proof-tree} $A$ (or rather, $\textit{COPY}$ ?), having the \textit{wff} $A$ as an open assumption, concludes $A$ and is decorated by $x$ in the environment $\Gamma, x : A$.

“copy” = “use” ?
Natural deduction as a Typing System

- \( \Rightarrow \)-introduction:

\[
\begin{align*}
\Gamma, x : A & \vdash M : B \\
\Gamma & \vdash (\lambda x : A.M) : A \Rightarrow B
\end{align*}
\]

NB: In Coq’s concrete syntax

\[
\begin{align*}
\Gamma, x : A & \vdash M : B \\
\Gamma & \vdash (\text{fun } x : A \Rightarrow M) : A \rightarrow B
\end{align*}
\]

In fact, no need for \textsf{imp_i}:

\textsf{imp_i} A B (fun x:A=>M) is \textsf{fun} x:A=>M

This is \textit{(Creation of) unnamed function} (see next slide)
Natural deduction as a Typing System

- CoQ's definitions: Let myfavoritename := myterm.
  or Definition myfavoritename := myterm.

- Hence, think of Let myfunction := fun x:A => M. as
  myfunction (x:A) {
    ...
    M
    ...
  }

  It is of type A→B (if M is of type B)

  x is only available in the body M of the function.

  A is an assumption temporarily made for the (sub-)proof M of B.
Natural deduction as a Typing System

- \( \Rightarrow \)-elimination (a.k.a. *Modus Ponens*):

\[
\frac{\Gamma \vdash M : A \Rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M \, N : B}
\]

Again, no need for \( \text{imp}_{\, \land} \): \( \text{imp}_{\, \land} \ A \ B \ M \ N \) is \( M \, N \)

This is **Function application**
Natural deduction as a Typing System

• ∨-introduction:

\[
\begin{align*}
\Gamma & \vdash M : A \\
\Gamma & \vdash \text{or\_introl } M : A \lor B \\
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash M : B \\
\Gamma & \vdash \text{or\_intror } M : A \lor B \\
\end{align*}
\]

Again, no need for or\_il: \text{or\_il } A B M is \text{or\_introl } M
and similarly for or\_ir.

This is a Cast in a union type, with a tag to remember which side of
the union a term comes from.
Natural deduction as a Typing System

- ∨-elimination:

\[ \Gamma \vdash M : A \lor B \quad \Gamma, x : A \vdash N : C \quad \Gamma, y : B \vdash P : C \]

\[ \Gamma \vdash \text{match } M \text{ with } \begin{array}{c} \text{or_introl } x \Rightarrow N \ \text{or_intror } y \Rightarrow P \end{array} \text{ end} \]

This is a **Case analysis** on the tag specifying which part of the union \( M \) comes from.

\( \text{or_e } A \ B \ M \ (\text{fun } x:A=>N) \ (\text{fun } y:A=>P) \)

\( \text{match } M \text{ with } \begin{array}{c} \text{or_introl } x \Rightarrow N \ \text{or_intror } y \Rightarrow P \end{array} \text{ end} \)
Natural deduction as a Typing System

- ∧-introduction:

\[ \Gamma \vdash M : A \quad \Gamma \vdash N : B \quad \Gamma \vdash M : A \quad \Gamma \vdash N : B \]

\[ \Gamma \vdash \text{conj} \; M \; N : A \land B \quad \Gamma \vdash \text{pair} \; M \; N : A \ast B \]

\text{pair} M N \text{ abbreviated as } (M, N)

Again, no need for \text{and}_i: \text{and}_i A \; B \; M \; N \text{ is conj}\; M \; N

This is a Pair / 2-component Structure.
Natural deduction as a Typing System

• $\land$-elimination:

\[
\Gamma \vdash M : A \land B
\]

\[
\begin{align*}
\text{match } M \text{ with} \\
\Gamma \vdash \text{conj } x \ y \Rightarrow x : A \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{match } M \text{ with} \\
\Gamma \vdash \text{conj } x \ y \Rightarrow y : B \\
\text{end}
\end{align*}
\]

This is the Access to one component of a Pair / binary structure.
Natural deduction as a Typing System

- Point raised: Type “Inference”

- With these rules, it is true that

  Given $\Gamma, M$, there is at most one type $A$ s.t. there is a decorated proof-tree concluding $\Gamma \vdash M : A$.

  (& there is an algorithm to find it: run by Check command in CoQ)

- Exercise: check this fact in all the decorated rules.

  In particular, it relies on indicating “: $A$” in $\rightarrow$-intro rule $(\lambda x : A. M)$

  Otherwise, how would you find the type of $\lambda x.x$?

  Exercise: why is it not needed in the discharges of $\lor$-elim?
Natural deduction: the actual rules for ⊥ and ¬

• So far, no computational interpretation of ⊥ and ¬:

• ⊥-introduction: none

\[
\begin{array}{c}
\top \\
\vdots \\
\neg A
\end{array}
\]

• ¬-introduction: [A]

\[
\begin{array}{c}
\top \\
\vdots \\
\neg A
\end{array}
\]

• ¬-elimination: A, ¬A

\[
\begin{array}{c}
A \\
\vdots \\
\neg A
\end{array}
\]

⊥-elimination: ⊥

\[
\begin{array}{c}
\top \\
\vdots \\
A
\end{array}
\]
Natural deduction as a Typing System for $\lambda$-calculus

- Implicational fragment ($\Rightarrow$ as the only connective), in intuitionistic logic. Syntax obtained:

\[ M, N, P, \ldots ::= x \mid \lambda x : A. M \mid M \ N \]

is the $\lambda$-calculus (Church, 30’s).

Set of variables decorating open assumptions (k.a. Free variables):

\[
\text{FV}(x) = x \\
\text{FV}(M \ N) = \text{FV}(M) \cup \text{FV}(N) \\
\text{FV}(\lambda x : A. M) = \text{FV}(M) \setminus \{x\}
\]

Full lecture on it later.
Questions?