UNIVERSITY OF CAMBRIDGE

Introductory Logic

Lecture 8: Natural Deduction, Intuitionism

Alan Mycroft

Computer Laboratory, University of Cambridge, UK
http://www.cl.cam.ac.uk/~am21

MPhil in ACS – 2011/12

Lecture Outline

- Natural Deduction
- Intuitionism
- Proofs as programs, formulae as types

Natural Deduction

Previously we gave Hilbert-style proof rules for FOL: a set of axioms with a single inference rule (modus ponens).
An alternative sound and complete proof system can be given in *natural deduction* style. Here judgements are of the form $\Gamma \vdash \tau$ as previously, but the key differences are:

- there are two forms of rules for each logical connective (*introduction* rules and *elimination* rules).
- rules exploit the ability to vary Γ.

Natural deduction proofs have neat connections to programs and many type systems are specified in natural deduction style.

Natural Deduction Rules

$$\text{(AND-I)}\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi} \qquad \text{(AND-E1)}\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \qquad \text{(AND-E2)}\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$$

$$\text{(OR-I1)}\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \text{(OR-I2)}\frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$$

$$\text{(OR-E)}\frac{\Gamma \vdash \phi \vee \psi \quad \Gamma, \phi \vdash \tau \quad \Gamma, \psi \vdash \tau}{\Gamma \vdash \tau}$$

$$\text{(IMP-I)}\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi} \qquad \text{(IMP-E)}\frac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

$$\text{(NOT-I)}\frac{\Gamma, \phi \vdash \bot}{\Gamma \vdash \neg\phi} \qquad \text{(NOT-E)}\frac{\Gamma \vdash \phi \quad \Gamma \vdash \neg\phi}{\Gamma \vdash \psi}$$

Natural Deduction Rules (2)

$$\text{(ASSUME)}\frac{}{\Gamma, \phi \vdash \phi}$$

$$\text{(ALL-I)}\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall x\ \phi}\ \text{provided } x \text{ is not free in } \Gamma \qquad \text{(ALL-E)}\frac{\Gamma \vdash \forall x\ \phi}{\Gamma \vdash \phi[t/x]}$$

$$\text{(DNEG)}\frac{\Gamma \vdash \neg\neg\phi}{\Gamma \vdash \phi}$$

Natural Deduction Rules (3)

It turns out that these above rules R, we once again have soundness and completeness:

$$\Gamma \vdash_R \phi \text{ iff } \Gamma \models \phi$$

Without the rule (DNEG) the rules are incomplete for FOL, but allow an additional *constructive* or *intuitionistic* reading. For more details see

`http://plato.stanford.edu/entries/logic-intuitionistic/`

but in essence a sentence like $A \vee \neg A$ ("Law of the Excluded Middle") is only provable when we can first establish A or establish $\neg A$. This anticipates Gödel's idea of statements which are neither true nor false, and indeed to computer science where a proof search procedure may return "yes", "no" or merely fail to terminate.

Intuitionism is weaker?

One might argue that intuitionistic proof systems are weaker since they lose completeness by dropping DNEG. In terms of provability this is certainly true.
However, in terms of expressivity, at some level they include classical logic as a special case. Glivenko's theorem states (for the propositional subset of FOL):

Theorem

If Γ is a set of propositional formulas and ϕ a propositional formula, then $\Gamma \vdash \phi$ using classical logic if and only if $\Gamma \vdash \neg\neg\phi$ using intuitionistic logic.

See e.g.

`http://en.wikipedia.org/wiki/Gödel-Gentzen_negative_translation`

Intuitionism and Programming Languages

One reason why intuitionistic reason is relevant to Computer Science is the Curry-Howard correspondence relating formulae to types and proofs to programs.

We consider a language like ML or lambda-calculus with syntax (note we use the word 'sum' for 'disjoint union'):

e	$::=$	$x \mid \lambda x.e \mid e_1\ e_2$	functions
	$\mid$	$(e_1, e_2) \mid \pi_1 e \mid \pi_2 e$	pairing, projection
	$\mid$	$in_1 e \mid in_2 e$	inject to sum
	$\mid$	$case\ e_0\ of\ \{in_1 x \Rightarrow e_1; in_2 y \Rightarrow e_2\}$	case analysis of sum

Typed lambda-calculus

In programming languages type correctness of these rules are given by natural deduction-style rules. Types have syntax (assuming a primitive type *int*):

$$t ::= int \mid t + t' \mid t \times t' \mid t \to t'$$

Judgements are $\Gamma \vdash e : t$ ('in context Γ we infer that e has type t') where context Γ is a set of assumptions of the form $x : t$. We also need notion to represent scoping: ignore any existing assumptions for x in Γ and add $x : t$:

$$\Gamma[x : t] = \Gamma \setminus \{x : t' \mid \exists t'(x : t' \in \Gamma)\} \cup \{x : t\}$$

The rules are naturally given as follows:

Typed lambda-calculus (2)

$$(\text{VAR})\frac{}{\Gamma[x : t] \vdash x : t} \qquad (\times\text{-I})\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e_1, e_2) : t_1 \times t_2}$$

$$(\times\text{-E1})\frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash \pi_1 e : t_1} \qquad (\times\text{-E2})\frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash \pi_2 e : t_2}$$

$$(+\text{-I1})\frac{\Gamma \vdash e : t_1}{\Gamma \vdash in_1 e : t_1 + t_2} \qquad (+\text{-I2})\frac{\Gamma \vdash e : t_2}{\Gamma \vdash in_2 e : t_1 + t_2}$$

$$(+\text{-E})\frac{\Gamma \vdash e_0 : t_1 + t_2 \quad \Gamma[x : t_1] \vdash e_1 : t_3 \quad \Gamma[y : t_2] \vdash e_2 : t_3}{\Gamma \vdash case\ e_0\ of\ \{in_1 x \Rightarrow e_1; in_2 y \Rightarrow e_2\} : t_3}$$

$$(\to\text{-I})\frac{\Gamma[x : t_1] \vdash e : t_2}{\Gamma \vdash \lambda x.e : t_1 \to t_2} \qquad (\to\text{-E})\frac{\Gamma \vdash e_1 : t_2 \to t_3 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash e_1\ e_2 : t_3}$$

Curry-Howard Correspondence

Note the correspondence between the two systems. If we erase all uses of x and e (replacing $e : t$ in judgement forms, including Γ), and similarly only consider the propositional forms involving AND, OR and IMPLIES, then intuitionistic propositional deduction and type checking are isomorphic (replacing $\{\wedge, \vee, \to\}$ with $\{\times, +, \to\}$.

Hence this view is often called 'propositions as types'. But intuitionism gives us more: the x and e parts of the judgments can be seen as *proof forms* for intuionistic proofs, hence the enriched view (of 'propositions as types'): 'programs as proofs'.

It's worth reading this case carefully: a proof of $A \wedge B$ is a proof of A and a proof of B; a proof of $A \vee B$ is a proof of A or a proof of B *and* a marker as to which one has been proved. A proof of $A \to B$ is a function which takes a proof of A and returns a proof of B.