Logical and Meta-Logical Frameworks

Frank Pfenning

Marktoberdorf Summer School 2001
July 25-August 4, 2001

First Things First

- If you play squash see me after lecture!

Outline of Four Lectures

- **Lecture 1**: Higher-Order Abstract Syntax
- **Lecture 2**: Judgments as Types
- **Lecture 3**: Proof Search and Representation
- **Lecture 4**: Meta-Logical Frameworks

Logical and Meta-Logical Frameworks
Lecture 1: Higher-Order Abstract Syntax

1. Introduction
2. Parametric and hypothetical judgments
3. Higher-order abstract syntax
4. Properties of representations

Deductive Systems

- Judgment — object of knowledge
- Evident Judgment — something we know
- Deduction — evidence for a judgment
- Basic Judgments, for example
  - P is a proposition (P *prop*)
  - P is true (P *true*)
- Judgment Forms, for example
  - Parametric judgments x *term* $\vdash P(x) \supset Q(x)$ *prop*
  - Hypothetical judgments P *true*, $(P \supset Q)$ *true* $\vdash Q$ *true*
- Following Martin-Löf ['83,'85,'96]

Examples of Deductive Systems

- From logic
  - Natural deduction P_1 *true*$, \ldots, P_n$ *true* $\vdash Q$ *true*
  - Sequent calculus P_1 *hyp*$, \ldots, P_n$ *hyp* $\vdash Q$ *true*
  - Axiomatic derivation $\vdash Q$ *valid*
- Other logics (temporal, modal, linear, higher-order, dynamic, non-commutative, belief, relevance, ...)
- From programming languages
  - Typing $x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash e : \tau$
  - Evaluation $e \hookrightarrow v$
  - Equivalence $x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash e_1 \simeq e_2 : \tau$
  - Compilation $x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash e \rightarrow c$

Logical Frameworks

- Logical Framework — meta-language for deductive systems
- Tasks
  - Specification of abstract syntax and rules
  - Representation and verification of deductions
  - Implementation of algorithms (search, type inference)
- Applications
  - Reasoning in logical systems [Nipkow]
  - Verification (hardware, software, protocols) [Constable] [Grumberg]
  - Proof-carrying code [Necula]
  - Education
- Factor implementation effort!

Examples of Logical Frameworks

- Hereditary Harrop formulas
  Isabelle, λProlog
- λ^{Π} type theory
  Automath, LF, Elf, Twelf
- Substructural logics and type theories
  Forum, Linear LF, Ordered LF, Ludics(?) [Girard]
- Equational logic and rewriting
  Maude, ELAN, labelled deductive systems
- Constructive type theories
  ALF, Agda, Coq, LEGO, Nuprl

Meta-Logical Frameworks

- Meta-Logical Framework —
  meta-language for reasoning **about** deductive system
- Tasks
  - Specification of abstract syntax and rules
  - Proof of properties of deductive systems
- Applications
  - Logic specification and verification
  - Programming language design
  - Reflection and proof compression

Examples of Meta-Logical Frameworks

- Finitary inductive definitions
  FS_0 [Feferman'88]

- Definitional reflection
  $FOL^{\Delta N}$ [McDowell&Miller'97]

- Higher-level judgments and regular worlds
  M_2, Twelf [Schürmann'00]

- Other systems used as meta-logical frameworks
  - Constructive type theories
    Agda, Coq, LEGO, Nuprl
  - Higher-order logic
    HOL, Isabelle/HOL
  - Rewriting logic
    Maude

These Lectures

- Running examples: natural deduction, axiomatic derivations
- Logical framework: LF, Elf
- Meta-logical framework: Twelf
- Reference:

  Logical frameworks.
  Handbook of Automated Reasoning,
  Chapter 16, pp. 977-1061,
  Elsevier Science and MIT Press, June 2001.

- Textbook:

  Computation and Deduction.
  Cambridge University Press, Fall 2001.

- Implementation: twelf.org

Terms and Propositions of First-Order Logic

- Basic judgments: t *term*, P *prop*
- Parametric judgments:

$$x_1 \textit{ term}, \ldots, x_n \textit{ term} \vdash t \textit{ term}$$

$$x_1 \textit{ term}, \ldots, x_n \textit{ term} \vdash P \textit{ prop}$$

- x_i are parameters
- x_i *term* are hypotheses
- Notation: $\Delta = x_1 \textit{ term}, \ldots, x_n \textit{ term}$
- Assume all x_i distinct!

Substitution

- Defines meaning of parametric judgment
- Substitution $[t/x]s$ and $[t/x]P$ (defined as usual)
- Substitution property (similarly for propositions):

  If $\Delta, x\ \textit{term}, \Delta' \vdash s\ \textit{term}$
  and $\Delta \vdash t\ \textit{term}$
  then $\Delta, \Delta' \vdash [t/x]s\ \textit{term}$

- Hypothesis rule:

$$\frac{}{\Delta, x\ \textit{term}, \Delta' \vdash x\ \textit{term}}\ \mathsf{hyp}$$

- Parameters need not be used (weakening)
- Parameters may be used more than once (contraction)

Logical Connectives

- Implication formation

$$\frac{\Delta \vdash P \textit{ prop} \qquad \Delta \vdash Q \textit{ prop}}{\Delta \vdash P \supset Q \textit{ prop}} \supset F$$

- Negation formation

$$\frac{\Delta \vdash P \textit{ prop}}{\Delta \vdash \neg P \textit{ prop}} \neg F$$

- Universal quantification

$$\frac{\Delta, x \textit{ term} \vdash P \textit{ prop}}{\Delta \vdash \forall x.\, P \textit{ prop}} \forall F$$

Free and Bound Variables

- Free variables defined as usual
- Bound variables defined as usual (binder $\forall x$)
- $\forall x.\, P = \forall y.\, [y/x]P$ provided y not free in P
- Identify propositions up to renaming of bound variables
- Substitution avoids capture by silent renaming, e.g.,

$$
\begin{array}{rcl}
[y/x](\forall y.\, P\ y\ x) & = & [y/x](\forall y'.\, P\ y'\ x) \\
 & = & \forall y'.\, P\ y'\ y \\
[y/x](\forall y.\, P\ y\ x) & \neq & \forall y.\, P\ y\ y
\end{array}
$$

- Parameters in context x_1 *term*$, \ldots, x_n$ *term* are all distinct

Predicate and Function Symbols

- Predicate symbols p^n of arity n
- Functions symbols f^n of arity n
- "Uninterpreted" in first-order logic:
  judgments are parametric in p^n and f^n
- May be interpreted in arithmetic or other theories:
  judgments are no longer parametric

Representing Terms and Propositions

- Two critical issues:
  - How to represent variables and substitution
  - How to represent judgments t *term* and P *prop*
- Three standard variable techniques:
  - Named (string) representation
  - De Bruijn representation
  - **Higher-order abstract syntax**
- Two standard judgment techniques:
  - Judgments as propositions
  - **Judgments as types**

Simply-Typed Fragment of LF

- Meta-language: $\lambda^{\rightarrow}$ as fragment of LF

$$
\begin{array}{rrcl}
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}\mathit{type} \mid \Sigma, c{:}A \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A \\
\textit{Types} & A & ::= & a \mid A_1 \rightarrow A_2 \\
\textit{Objects} & M & ::= & c \mid x \mid \lambda x{:}A.\, M \mid M_1\, M_2
\end{array}
$$

- Type constants a, object constants c, object variables x
- Judgments defining meta-language $\lambda^{\rightarrow}$ (more later)
  - $\Sigma\ \mathit{sig}$ — signature Σ is valid
  - $\Gamma\ \mathit{ctx}$ — context Γ is valid
  - $\vdash_\Sigma A : \mathit{type}$ — type A is a valid
  - $\Gamma \vdash_\Sigma M : A$ — object M has type A

Representation of Terms

- Introduce type i for terms

$$\mathsf{i} : \mathit{type}$$

- Property: if t *term* then $\ulcorner t \urcorner : \mathsf{i}$
- More generally:

  If x_1 *term*$, \ldots, x_n$ *term* $\vdash t$ *term*
  then $x_1{:}\mathsf{i}, \ldots, x_n{:}\mathsf{i} \vdash \ulcorner t \urcorner : \mathsf{i}$

- Representing **parameters as parameters** in LF,

$$\ulcorner x \urcorner = x$$

- Representing **hypotheses as hypotheses** in LF,

$$\ulcorner x_1 \mathit{term}, \ldots, x_n \mathit{term} \urcorner \;=\; x_1{:}\mathsf{i}, \ldots, x_n{:}\mathsf{i}$$

Representation of Propositions

- Introduce type o for propositions

$$\mathsf{o} : \mathit{type}$$

- Property: if P *prop* then $\ulcorner P \urcorner : \mathsf{o}$
- More generally:

  If x_1 *term*, $\ldots$, x_n *term* $\vdash P$ *prop*
  then $x_1{:}\mathsf{i}, \ldots, x_n{:}\mathsf{i} \vdash \ulcorner P \urcorner : \mathsf{o}$

- Again: parameters as parameters, hypotheses as hypotheses

Constructors as Constants, Implication

- Implication

$$\frac{\Delta \vdash P \; \textit{prop} \qquad \Delta \vdash Q \; \textit{prop}}{\Delta \vdash P \supset Q \; \textit{prop}} \supset F$$

$$\ulcorner P \supset Q \urcorner = \mathsf{imp} \, \ulcorner P \urcorner \, \ulcorner Q \urcorner$$

$$\mathsf{imp} : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$$

Constructors as Constants, Negation

- Negation

$$\frac{\Delta \vdash P \; prop}{\Delta \vdash \neg P \; prop} \neg F$$

$$\ulcorner \neg P \urcorner = \mathsf{not} \ulcorner P \urcorner$$

$$\mathsf{not} : \mathsf{o} \rightarrow \mathsf{o}$$

Constructors as Constants, Universal Quantification

- Universal quantification

$$\frac{\Delta, x\ \mathit{term} \vdash P\ \mathit{prop}}{\Delta \vdash \forall x.\, P\ \mathit{prop}}\ \forall F$$

$$\ulcorner \forall x.\, P \urcorner = \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)$$

$$\mathsf{forall} : (\mathsf{i} \to \mathsf{o}) \to \mathsf{o}$$

- Essential reasoning

$$\frac{\ulcorner \Delta \urcorner \vdash \mathsf{forall} : (\mathsf{i} \to \mathsf{o}) \to \mathsf{o} \qquad \dfrac{\ulcorner \Delta \urcorner, x{:}\mathsf{i} \vdash \ulcorner P \urcorner : \mathsf{o}}{\ulcorner \Delta \urcorner \vdash \lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner : \mathsf{i} \to \mathsf{o}}}{\ulcorner \Delta \urcorner \vdash \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner) : \mathsf{o}}$$

- **Bound variables as λ-bound variables** in LF

Function and Predicate Symbols

- Propositional or term constants have arity 0.
- For function symbols f^n:

$$\ulcorner f^n(t_1, \ldots, t_n) \urcorner = \mathsf{f}\ \ulcorner t_1 \urcorner\ \ldots\ \ulcorner t_n \urcorner$$

$$\mathsf{f} : \underbrace{\mathsf{i} \rightarrow \cdots \mathsf{i} \rightarrow}_{n} \mathsf{i}$$

- For predicate symbols p^n:

$$\ulcorner p^n(t_1, \ldots, t_n) \urcorner = \mathsf{p}\ \ulcorner t_1 \urcorner\ \ldots\ \ulcorner t_n \urcorner$$

$$\mathsf{p} : \underbrace{\mathsf{i} \rightarrow \cdots \mathsf{i} \rightarrow}_{n} \mathsf{o}$$

- Status as parameters (in context Δ) or constants (in signature Σ) depends on application

Examples of Representations

- Represent predicate parameters by corresponding LF parameters

- $\ulcorner P \supset (Q \supset P) \urcorner = \mathsf{imp}\ P\ (\mathsf{imp}\ Q\ P)$
  for $P : \mathsf{o}, Q : \mathsf{o}$

- $\ulcorner \forall x.\, P(x) \supset Q(x) \urcorner = \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \mathsf{imp}\ (P\ x)\ (Q\ x))$
  for $P : \mathsf{i} \to \mathsf{o}, Q : \mathsf{i} \to \mathsf{o}$

- $\ulcorner \forall x.\, P \supset Q(x) \urcorner = \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \mathsf{imp}\ P\ (Q\ x))$
  for $P : \mathsf{o}, Q : \mathsf{i} \to \mathsf{o}$
  Note: substituent for P cannot refer to x

Summary of Representation

- Terms and propositions

$$
\begin{array}{lcl}
\ulcorner P \supset Q \urcorner & = & \mathsf{imp}\,\ulcorner P \urcorner\,\ulcorner Q \urcorner \\
\ulcorner \neg P \urcorner & = & \mathsf{not}\,\ulcorner P \urcorner \\
\ulcorner \forall x.\,P \urcorner & = & \mathsf{forall}\,(\lambda x{:}\mathsf{i}.\,\ulcorner P \urcorner)
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{i} & : & \mathit{type} \\
\mathsf{o} & : & \mathit{type} \\
\mathsf{imp} & : & \mathsf{o} \to \mathsf{o} \to \mathsf{o} \\
\mathsf{not} & : & \mathsf{o} \to \mathsf{o} \\
\mathsf{forall} & : & (\mathsf{i} \to \mathsf{o}) \to \mathsf{o}
\end{array}
$$

- Variables are represented as variables
  Higher-order abstract syntax

- Variable renaming as α-conversion in LF

- Essentially **open-ended** [Constable]

Adequacy Theorem for Propositions

- With respect to fixed signature (suppressed)
- Validity:

  If $\Delta \vdash P$ *prop then* $\ulcorner\Delta\urcorner \vdash \ulcorner P\urcorner : \mathsf{o}$

- Injectivity: *If* $\ulcorner P\urcorner = \ulcorner Q\urcorner$ *then* $P = Q$
- Surjectivity?

  If $\ulcorner\Delta\urcorner \vdash M : \mathsf{o}$
  then $M = \ulcorner P\urcorner$ *for some* P *with* $\Delta \vdash P$ *prop?*

- Compositionality:

  $$[\ulcorner t\urcorner/x]\ulcorner P\urcorner = \ulcorner [t/x]P\urcorner$$

Surjectivity

- Validity, injectivity, and compositionality by easy inductions
- Surjectivity fails:
  - Counterexample, for $\mathsf{p} : \mathsf{i} \to \mathsf{o}$

    $$\vdash \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\ ((\lambda q{:}\mathsf{o}.\ q)\ (p\ x))) : \mathsf{o}$$

    is not in the image of $\ulcorner\_\urcorner$
  - Solution: β-reduction to

    $$\vdash \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\ p\ x)$$

  - Counterexample, for $\mathsf{p} : \mathsf{i} \to \mathsf{o}$

    $$\vdash \mathsf{forall}\ \mathsf{p} : \mathsf{o}$$

    is not in the image of $\ulcorner\_\urcorner$
  - Solution: η-expansion to

    $$\vdash \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\ p\ x)$$

Definitional Equality for LF

- Equip LF with a notion of definitional equality
- $\Gamma \vdash_{\Sigma} M = N : A$ — objects M and N are definitionally equal
- Congruence generated from β- and η-conversion

$$\begin{aligned}(\lambda x{:}A.\, M)\ N &= [N/x]M \\ M{:}A \to B &= \lambda x{:}A.\, M\ x \quad \text{provided } x \text{ not free in } M\end{aligned}$$

- Define so that $\Gamma \vdash_{\Sigma} M = N : A$ ensures $\Gamma \vdash_{\Sigma} M : A$ and $\Gamma \vdash_{\Sigma} N : A$

Surjectivity Corrected

- Surjectivity (corrected):

  If $\ulcorner\Delta\urcorner \vdash M : \mathsf{o}$
  then $\ulcorner\Delta\urcorner \vdash M = \ulcorner P\urcorner : \mathsf{o}$
  for some P *with* $\Delta \vdash P$ *prop*

- Injectivity (retained):

  If $\ulcorner\Delta\urcorner \vdash \ulcorner P\urcorner = \ulcorner Q\urcorner : \mathsf{o}$
  then $P = Q$ *for* $\Delta \vdash P$ *prop and* $\Delta \vdash Q$ *prop*

- Recall: everything modulo renaming of bound variables

- Proofs via canonical forms

Canonical Forms

- $\Gamma \vdash_{\Sigma} M \Downarrow A$ — M is canonical of type A
- Intuition: canonical is β-normal and η-long:

  $$M \Downarrow A_1 \to \ldots \to A_k \to a$$

  iff

  $$M = \lambda x_1{:}A_1. \ldots \lambda x_k{:}A_k. h\ M_1\ \ldots\ M_n$$

  for a variable or constant h, type constant a, and canonical $M_1, \ldots, M_n$
- More formal definition later
- **Theorem:** Every valid object has an unique, equivalent canonical form
- Obtained by β-reduction and η-expansion

Injectivity Interpreted

- Recall injectivity:

  If $\ulcorner\Delta\urcorner \vdash \ulcorner P\urcorner = \ulcorner Q\urcorner : \mathsf{o}$
  then $P = Q$ *for every* $\Delta \vdash P$ *prop and* $\Delta \vdash Q$ *prop*

- No ambiguity in representation
- Stronger than usual in data representation:
  data type = representation type + equivalence relation
- Operations on objects well defined (coherence)
- Sometimes sacrificed, e.g.,
  integers $\ulcorner i\urcorner = \mathsf{diff}\ n\ m$ for n, m:nat with $i = n - m$

Surjectivity Interpreted

- Recall surjectivity:

  If $\ulcorner \Delta \urcorner \vdash M : \mathsf{o}$
  then $\ulcorner \Delta \urcorner \vdash M = \ulcorner P \urcorner : \mathsf{o}$
  for some P *with* $\Delta \vdash P$ *prop*

- No "junk" in representation type
- Stronger than usual in data representation:
  data structure = data type + invariants
- Incorporate invariants when possible
- Not always feasible, e.g.,
  linear λ-terms = λ-terms + linearity

Compositionality Interpreted

- Recall compositionality:

$$[\ulcorner t \urcorner / x] \ulcorner P \urcorner = \ulcorner [t/x] P \urcorner$$

- Representation commutes with substitution
- Consequence of representing variables as variables
- Substitution represented by β-reduction in LF, e.g.,

$$\ulcorner \forall x.\, P \urcorner = \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)$$
$$\ulcorner [t/x] P \urcorner = [\ulcorner t \urcorner / x] \ulcorner P \urcorner =_\beta (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)\ t$$

- Critical advantage of higher-order abstract syntax

Summary of Lecture 1

- Introduction and overview
- Parametric and hypothetical judgments, defined by substitution property
- Sample object language is first-order logic
- Meta-language is simply-typed fragment of LF
- Representation via higher-order abstract syntax
  - Variables as variables in LF
  - Variable renaming as α-conversion in LF
  - Substitution as β-conversion in LF
- Representation is injective, surjective, compositional

Preview of Lecture 2: Judgments as Types

1. Natural Deduction

2. Judgments as Types

3. Dependent Function Types in LF

4. Representing Parametric and Hypothetical Judgments

Reminder

- If you play squash see me now!

Logical and Meta-Logical Frameworks
Lecture 2: Judgments as Types

1. Natural Deduction
2. Judgments as Types
3. Dependent Function Types in LF
4. Representing Parametric and Hypothetical Judgments

Review of Lecture 1: Higher-Order Abstract Syntax

- Meta-language: simply-typed λ-calculus as fragment of LF
- Representing terms and proposition

$$
\begin{array}{lcl}
\ulcorner P \supset Q \urcorner & = & \mathsf{imp}\,\ulcorner P \urcorner\,\ulcorner Q \urcorner \\
\ulcorner \neg P \urcorner & = & \mathsf{not}\,\ulcorner P \urcorner \\
\ulcorner \forall x.\, P \urcorner & = & \mathsf{forall}\,(\lambda x{:}\mathsf{i}.\,\ulcorner P \urcorner)
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{i} & : & \mathit{type} \\
\mathsf{o} & : & \mathit{type} \\
\mathsf{imp} & : & \mathsf{o} \to \mathsf{o} \to \mathsf{o} \\
\mathsf{not} & : & \mathsf{o} \to \mathsf{o} \\
\mathsf{forall} & : & (\mathsf{i} \to \mathsf{o}) \to \mathsf{o}
\end{array}
$$

- Variables represented as variables in LF
- Variable renaming via α-conversion in LF
- Definitional equality in LF generated from $\beta\eta$-conversion
- Adequacy: representation is **compositional bijection**

$$\ulcorner [t/x]s \urcorner = [\ulcorner t \urcorner / x]\ulcorner s \urcorner, \quad \ulcorner [t/x]P \urcorner = [\ulcorner t \urcorner / x]\ulcorner P \urcorner$$

Natural Deduction

- Basic judgment: P *true*, presupposing P *prop*
- Intuitively: P *has a verification* [Martin-Löf'83,'96]
- Parametric and hypothetical judgment $\Delta \vdash P$ *true*
- Need hypotheses
  - x *term* for term parameter x (for $\forall$)
  - p *prop* for propositional parameter p (for $\neg$)
  - $u{:}Q$ *true* for proposition Q and proof parameter u (for $\supset$)
- Hypothesis rule

$$\frac{}{\Delta, u{:}P\ \mathit{true}, \Delta' \vdash P\ \mathit{true}}\, u$$

Substitution Principles

- Recall: meaning of parametric judgments
- More complicated than before, because hypotheses may contain parameters (Δ has internal dependencies)
- Example: x *term*, $u{:}P(x)$ *true* $\vdash P(x)$ *true*
- For term parameters (similarly for propositional parameters)

  If Δ, x *term*, $\Delta' \vdash P$ *true*
  and $\Delta \vdash t$ *term*
  then $\Delta, [t/x]\Delta' \vdash [t/x]P$ *true*

- For proof parameters

  If $\Delta, u{:}P$ *true*, $\Delta' \vdash Q$ *true*
  and $\Delta \vdash P$ *true*
  then $\Delta, \Delta' \vdash Q$ *true*

Introduction and Elimination Rules

- The meaning of a connective is given by the rule(s) for inferring it, the introduction rule(s)
- Corresponding elimination rule(s) justified from introduction rule(s)
- Local soundness: we cannot gain information by an introduction followed by an elimination
- Local soundness is guaranteed by a local reduction
- Local completeness: we can recover the information in a connective by elimination(s)
- Local completeness is guaranteed by a local expansion
- For local completeness and expansion see [notes]

Truth of Implication

- Introduction rule:

$$\frac{\Delta, u{:}P\ \mathit{true} \vdash Q\ \mathit{true}}{\Delta \vdash P \supset Q\ \mathit{true}} \supset I^u$$

- Elimination rule:

$$\frac{\Delta \vdash P \supset Q\ \mathit{true} \qquad \Delta \vdash P\ \mathit{true}}{\Delta \vdash Q\ \mathit{true}} \supset E$$

- Local reduction (soundness of elimination rule)

$$\cfrac{\cfrac{\begin{array}{c}\mathcal{D}\\ \Delta, u{:}P\ \mathit{true} \vdash Q\ \mathit{true}\end{array}}{\Delta \vdash P \supset Q\ \mathit{true}} \supset I^u \qquad \begin{array}{c}\mathcal{E}\\ \Delta \vdash P\ \mathit{true}\end{array}}{\Delta \vdash Q\ \mathit{true}} \supset E \quad \longrightarrow \quad \begin{array}{c}[\mathcal{E}/u]\mathcal{D}\\ \Delta \vdash Q\ \mathit{true}\end{array}$$

by substitution principle for proofs

Truth of Negation

- Introduction rule:

$$\frac{\Delta, q\ \textit{prop}, u{:}P\ \textit{true} \vdash q\ \textit{true}}{\Delta \vdash \neg P\ \textit{true}}\ \neg I^{q,u}$$

- Note propositional parameter q
- Elimination rule:

$$\frac{\Delta \vdash \neg P\ \textit{true} \qquad \Delta \vdash P\ \textit{true}}{\Delta \vdash Q\ \textit{true}}\ \neg E$$

- Definition of logical connectives only via judgmental notions
- Orthogonality and open-endedness

Local Reduction for Negation

- Local reduction

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ \Delta, q\ \textit{prop}, u{:}P\ \textit{true} \vdash q\ \textit{true}\end{matrix}}{\Delta \vdash \neg P\ \textit{true}}\ \neg I^{q,u} \qquad \begin{matrix}\mathcal{E}\\ \Delta \vdash P\ \textit{true}\end{matrix}}{\Delta \vdash Q\ \textit{true}}\ \neg E$$

$$\longrightarrow \qquad \begin{matrix}[\mathcal{E}/u][Q/q]\mathcal{D}\\ \Delta \vdash Q\ \textit{true}\end{matrix}$$

- First substitution for proposition q

$$\begin{matrix}[Q/q]\mathcal{D}\\ \Delta, u{:}P\ \textit{true} \vdash Q\ \textit{true}\end{matrix}$$

- Second substitution for proof u

$$\begin{matrix}[\mathcal{E}/u][Q/q]\mathcal{D}\\ \Delta \vdash Q\ \textit{true}\end{matrix}$$

Truth of Universal Quantification

- Introduction rule:

$$\frac{\Delta, x\ \textit{term} \vdash P\ \textit{true}}{\Delta \vdash \forall x.\, P\ \textit{true}}\ \forall I$$

- Elimination rule:

$$\frac{\Delta \vdash \forall x.\, P\ \textit{true} \qquad \Delta \vdash t\ \textit{term}}{\Delta \vdash [t/x]P\ \textit{true}}\ \forall E$$

- Local reduction:

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta, x\ \textit{term} \vdash P\ \textit{true}\end{array}}{\Delta \vdash \forall x.\, P\ \textit{true}}\ \forall I \qquad \begin{array}{c}\mathcal{T}\\ \Delta \vdash t\ \textit{term}\end{array}}{\Delta \vdash [t/x]P\ \textit{true}}\ \forall E \quad \longrightarrow \quad \begin{array}{c}[t/x]\mathcal{D}\\ \Delta \vdash [t/x]P\ \textit{true}\end{array}$$

by substitution principle for terms

Representation of Deductions

- Represent **judgments as types** in LF (ignoring hyps.)

$$\ulcorner P\ \mathit{true} \urcorner = \mathsf{true}\ \ulcorner P \urcorner$$

$$\vdash \mathsf{true}\ \ulcorner P \urcorner : \mathit{type}$$

$$\mathsf{true} : \mathsf{o} \to \mathit{type}$$

- true is a type family indexed by objects of type o
- Represent **deductions as objects** in LF

$$\left\ulcorner \begin{array}{c} \mathcal{D} \\ P\ \mathit{true} \end{array} \right\urcorner = M \quad \text{such that} \quad \vdash M : \mathsf{true}\ \ulcorner P \urcorner$$

- Requires extension of simply-typed fragment of LF

Representation of Inference Rules as Constants

- Example: implication elimination (ignoring Δ)

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta \vdash P \supset Q\ \mathit{true}\end{array} \qquad \begin{array}{c}\mathcal{E}\\ \Delta \vdash P\ \mathit{true}\end{array}}{\Delta \vdash Q\ \mathit{true}} \supset E \urcorner = \mathsf{impe}\, \ulcorner \mathcal{D} \urcorner\, \ulcorner \mathcal{E} \urcorner$$

- Translation into LF (ignoring Δ)

$$\dfrac{\begin{array}{rcl}\ulcorner \mathcal{D} \urcorner & : & \mathsf{true}\, (\mathsf{imp}\, \ulcorner P \urcorner\, \ulcorner Q \urcorner)\\ \ulcorner \mathcal{E} \urcorner & : & \mathsf{true}\, \ulcorner P \urcorner\end{array}}{\mathsf{impe}\, \ulcorner \mathcal{D} \urcorner\, \ulcorner \mathcal{E} \urcorner \ : \ \mathsf{true}\, \ulcorner Q \urcorner}$$

- Declaration for constant impe in LF

$$\mathsf{impe} : \mathsf{true}\, (\mathsf{imp}\, \ulcorner P \urcorner\, \ulcorner Q \urcorner) \rightarrow \mathsf{true}\, \ulcorner P \urcorner \rightarrow \mathsf{true}\, \ulcorner Q \urcorner$$

Schematic Rules

- Rules are schematic, e.g.,

$$\frac{\Delta \vdash P \supset Q \; \textit{true} \qquad \Delta \vdash P \; \textit{true}}{\Delta \vdash Q \; \textit{true}} \supset E$$

  is schematic in propositions P and Q.

- Representation is schematic, e.g.,

$$\mathsf{impe}_{P,Q} : \mathsf{true}\ (\mathsf{imp}\ P\ Q) \to \mathsf{true}\ P \to \mathsf{true}\ Q$$

  for any P:o, Q:o by adequacy for propositions

- Internalize schematic judgments in LF (read Π as "*Pi*")

$$\mathsf{impe} : \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{imp}\ P\ Q) \to \mathsf{true}\ P \to \mathsf{true}\ Q$$

Representing Schematic Judgments

- $\Pi x{:}A.\,B$ must be a *type*, e.g.,

$$\mathsf{impe} : \Pi P{:}\mathsf{o}.\,\Pi Q{:}\mathsf{o}.\,\mathsf{true}\;(\mathsf{imp}\;P\;Q) \to \mathsf{true}\;P \to \mathsf{true}\;Q$$

- Constant impe takes 4 arguments

an object $P : \mathsf{o}$	a proposition P
an object $Q : \mathsf{o}$	a proposition Q
an object $D : \mathsf{true}\;(\mathsf{imp}\;P\;Q)$	a deduction of $P \supset Q$ *true*
an object $E : \mathsf{true}\;P$	a deduction of P *true*
and constructs	
the object $\mathsf{impe}\;P\;Q\;D\;E : \mathsf{true}\;Q$	a deduction of Q *true*

Dependent Function Type in LF, Formation

- Dependent function type, formation

$$\frac{\Gamma \vdash A : \mathit{type} \qquad \Gamma, x{:}A \vdash B : \mathit{type}}{\Gamma \vdash \Pi x{:}A.\, B : \mathit{type}}\ \Pi F$$

$$\frac{\Gamma \vdash A : \mathit{type} \qquad \Gamma \vdash B : \mathit{type}}{\Gamma \vdash A \to B : \mathit{type}}\ {\to}F$$

- In $\Pi x{:}A.\, B$, x can occur in B

- Example:

$$\vdash \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{imp}\ P\ Q) \to \mathsf{true}\ P \to \mathsf{true}\ Q : \mathit{type}$$

- Different from **polymorphism** (not available in LF)

$$\vdash \Lambda\alpha{:}\mathit{type}.\, \lambda x{:}\alpha.\, x : \forall\alpha{:}\mathit{type}.\, \alpha \to \alpha$$

Dependent Function Type, Intro and Elim

- Dependent function type, introduction

$$\frac{\Gamma \vdash A : \mathit{type} \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : \Pi x{:}A.\, B}\ \Pi I$$

$$\frac{\Gamma \vdash A : \mathit{type} \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : A \to B}\ {\to}I$$

- Dependent function type, elimination

$$\frac{\Gamma \vdash M : \Pi x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : [N/x]B}\ \Pi E$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\ {\to}E$$

- Regard $A \to B$ as shorthand for $\Pi x{:}A.\, B$, where x not free in B

Representing Parametric Judgments

- Recall natural deduction judgment $\Delta \vdash P$ *true*
- Hypotheses Δ contain
  - x *term* for term parameter x (for $\forall$)
  - p *prop* for propositional parameter p (for $\neg$)
  - $u{:}Q$ *true* for proposition Q and proof parameter u (for $\supset$)
- Represent parameters as parameters in LF

$$
\begin{array}{rcl}
\ulcorner \cdot \urcorner & = & \cdot \\
\ulcorner \Delta, x\ \mathit{term} \urcorner & = & \ulcorner \Delta \urcorner, x{:}\mathsf{i} \\
\ulcorner \Delta, p\ \mathit{prop} \urcorner & = & \ulcorner \Delta \urcorner, p{:}\mathsf{o} \\
\ulcorner \Delta, u{:}Q\ \mathit{true} \urcorner & = & \ulcorner \Delta \urcorner, u{:}\mathsf{true}\,\ulcorner Q \urcorner
\end{array}
$$

Adequacy Theorem for Deductions, Bijection

- With respect to fixed signature (see later)
- Validity: *If* $\mathcal{D}$ *proves* $\Delta \vdash P$ *true then* $\ulcorner\Delta\urcorner \vdash \ulcorner\mathcal{D}\urcorner : \mathsf{true}\ \ulcorner P\urcorner$
- Injectivity:

  If $\ulcorner\Delta\urcorner \vdash \ulcorner\mathcal{D}\urcorner = \ulcorner\mathcal{E}\urcorner : \mathsf{true}\ \ulcorner P\urcorner$
  for $\mathcal{D}$ *and* $\mathcal{E}$ *proving* $\Delta \vdash P$ *true*
  then $\mathcal{D} = \mathcal{E}$ *(modulo variable renaming)*

- Surjectivity:

  If $\ulcorner\Delta\urcorner \vdash M : \mathsf{true}\ \ulcorner P\urcorner$
  then $\ulcorner\Delta\urcorner \vdash M = \ulcorner\mathcal{D}\urcorner : \mathsf{true}\ \ulcorner P\urcorner$
  for some $\mathcal{D}$ *proving* $\Delta \vdash P$ *prop*

Adequacy for Deductions, Compositionality

- Compositionality:

$$\begin{array}{llcl} \textit{Terms} & \ulcorner[t/x]\mathcal{D}\urcorner & = & [\ulcorner t\urcorner/x]\ulcorner\mathcal{D}\urcorner \\ \textit{Propositions} & \ulcorner[Q/p]\mathcal{D}\urcorner & = & [\ulcorner Q\urcorner/p]\ulcorner\mathcal{D}\urcorner \\ \textit{Proofs} & \ulcorner[\mathcal{E}/u]\mathcal{D}\urcorner & = & [\ulcorner\mathcal{E}\urcorner/u]\ulcorner\mathcal{D}\urcorner \end{array}$$

- Assume appropriate well-formedness for substitution, e.g.,

  $\mathcal{D}$ *proves* $\Delta, p\ \textit{prop}, \Delta' \vdash P\ \textit{true}$ *and* $\Delta \vdash Q\ \textit{prop}$
  so that $[Q/p]\mathcal{D}$ *proves* $\Delta, [Q/p]\Delta' \vdash [Q/p]P\ \textit{true}$

- Follows from the representation of variables as variables, hypotheses as hypotheses

Representing Uses of Hypotheses

- Hypothesis rule

$$\ulcorner \; \frac{}{\Delta, u{:}Q\ \mathit{true}, \Delta' \vdash Q\ \mathit{true}}\,u \; \urcorner$$

- Map to use of proof parameter in LF

$$\frac{}{\ulcorner\Delta\urcorner, u{:}\mathsf{true}\,\ulcorner Q\urcorner, \ulcorner\Delta'\urcorner \vdash u : \mathsf{true}\,\ulcorner Q\urcorner}$$

- Represent hypotheses as hypotheses
- Hypothesis labels u avoid ambiguity

Representation of Deductions, Implication Elim

- Implication elimination (review)

$$\ulcorner \dfrac{\dfrac{\mathcal{D}}{\Delta \vdash P \supset Q \ \mathit{true}} \qquad \dfrac{\mathcal{E}}{\Delta \vdash P \ \mathit{true}}}{\Delta \vdash Q \ \mathit{true}} \supset E \urcorner$$

$$\begin{array}{lcl}
\ulcorner\Delta\urcorner & \vdash & \ulcorner P \urcorner : \mathsf{o} \\
\ulcorner\Delta\urcorner & \vdash & \ulcorner Q \urcorner : \mathsf{o} \\
\ulcorner\Delta\urcorner & \vdash & \ulcorner \mathcal{D} \urcorner : \mathsf{true}\ (\mathsf{imp}\ \ulcorner P \urcorner\ \ulcorner Q \urcorner) \\
\ulcorner\Delta\urcorner & \vdash & \ulcorner \mathcal{E} \urcorner : \mathsf{true}\ \ulcorner P \urcorner \\
\hline
\ulcorner\Delta\urcorner & \vdash & \mathsf{impe}\ \ulcorner P \urcorner\ \ulcorner Q \urcorner\ \ulcorner \mathcal{D} \urcorner\ \ulcorner \mathcal{E} \urcorner : \mathsf{true}\ \ulcorner Q \urcorner
\end{array}$$

$$\mathsf{impe} \ : \ \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{imp}\ P\ Q) \rightarrow \mathsf{true}\ P \rightarrow \mathsf{true}\ Q$$

Representation of Deductions, Implication Intro

- Implication introduction

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D} \\ \Delta, u{:}P\ \mathit{true} \vdash Q\ \mathit{true}\end{array}}{\Delta \vdash P \supset Q\ \mathit{true}} \supset I^u \urcorner$$

$$\dfrac{\begin{array}{rcl} \ulcorner\Delta\urcorner & \vdash & \ulcorner P\urcorner : \mathsf{o} \\ \ulcorner\Delta\urcorner & \vdash & \ulcorner Q\urcorner : \mathsf{o} \\ \ulcorner\Delta\urcorner, u{:}\mathsf{true}\,\ulcorner P\urcorner & \vdash & \ulcorner\mathcal{D}\urcorner : \mathsf{true}\,\ulcorner Q\urcorner \end{array}}{\begin{array}{rcl} \ulcorner\Delta\urcorner & \vdash & \mathsf{impi}\,\ulcorner P\urcorner\,\ulcorner Q\urcorner\,(\lambda u{:}\mathsf{true}\,\ulcorner P\urcorner.\ulcorner\mathcal{D}\urcorner) \\ & & : \mathsf{true}\,(\mathsf{imp}\,\ulcorner P\urcorner\,\ulcorner Q\urcorner) \end{array}}$$

$$\mathsf{impi} : \Pi P{:}\mathsf{o}.\,\Pi Q{:}\mathsf{o}.\,(\mathsf{true}\ P \to \mathsf{true}\ Q) \to \mathsf{true}\ (\mathsf{imp}\ P\ Q)$$

- Critical step:

$$\dfrac{\ulcorner\Delta\urcorner, u{:}\mathsf{true}\,\ulcorner P\urcorner \vdash \ulcorner\mathcal{D}\urcorner : \mathsf{true}\,\ulcorner Q\urcorner}{\ulcorner\Delta\urcorner \vdash (\lambda u{:}\mathsf{true}\,\ulcorner P\urcorner.\ulcorner\mathcal{D}\urcorner) : (\mathsf{true}\,\ulcorner P\urcorner \to \mathsf{true}\,\ulcorner Q\urcorner)}$$

Representation of Deductions, Negation Intro

- Negation introduction

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta, q\ \textit{prop}, u{:}P\ \textit{true} \vdash q\ \textit{true}\end{array}}{\Delta \vdash \neg P\ \textit{true}}\neg I^{q,u} \urcorner$$

$$\dfrac{\begin{array}{rcl}\ulcorner\Delta\urcorner & \vdash & \ulcorner P\urcorner : \mathsf{o}\\ \ulcorner\Delta\urcorner, q{:}\mathsf{o}, u{:}\mathsf{true}\,\ulcorner P\urcorner & \vdash & \ulcorner\mathcal{D}\urcorner : \mathsf{true}\,\ulcorner q\urcorner\end{array}}{\begin{array}{rcl}\ulcorner\Delta\urcorner & \vdash & \mathsf{noti}\,\ulcorner P\urcorner\,(\lambda q{:}\mathsf{o}.\,\lambda u{:}\mathsf{true}\,\ulcorner P\urcorner.\,\ulcorner\mathcal{D}\urcorner)\\ & & :\mathsf{true}\,(\mathsf{not}\,\ulcorner P\urcorner)\end{array}}$$

$$\mathsf{noti} : \Pi P{:}\mathsf{o}.\,(\Pi q{:}\mathsf{o}.\,\mathsf{true}\ P \to \mathsf{true}\ q) \to \mathsf{true}\ (\mathsf{not}\ P)$$

- Critical step:

$$\dfrac{\ulcorner\Delta\urcorner, q{:}\mathsf{o}, u{:}\mathsf{true}\,\ulcorner P\urcorner \vdash \ulcorner\mathcal{D}\urcorner : \mathsf{true}\,\ulcorner q\urcorner}{\ulcorner\Delta\urcorner \vdash (\lambda q{:}\mathsf{true}.\,\lambda u{:}\mathsf{true}\,\ulcorner P\urcorner.\,\ulcorner\mathcal{D}\urcorner) : (\Pi q{:}\mathsf{true}.\,\mathsf{true}\,\ulcorner P\urcorner \to \mathsf{true}\,\ulcorner q\urcorner)}$$

Representation of Deductions, Negation Elim

- Negation elimination

$$\frac{\Delta \vdash \neg P \ \mathit{true} \qquad \Delta \vdash P \ \mathit{true}}{\Delta \vdash Q \ \mathit{true}} \neg E$$

- Development analogous to before (omitted)
- Representation

$$\mathsf{note} : \Pi P{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{not}\ P) \to \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ P \to \mathsf{true}\ Q$$

- Order of quantification over Q is irrelevant

Representation of Deductions, Universal Intro

- Recall $\ulcorner \forall x.\, P \urcorner = \mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)$

- Universal introduction

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta, x\ \mathit{term} \vdash P\ \mathit{true}\end{array}}{\Delta \vdash \forall x.\, P\ \mathit{true}}\,\forall I \urcorner$$

$$\dfrac{\begin{array}{lll}\ulcorner \Delta \urcorner, x{:}\mathsf{i} & \vdash & \ulcorner P \urcorner : \mathsf{o}\\ \ulcorner \Delta \urcorner, x{:}\mathsf{i} & \vdash & \ulcorner \mathcal{D} \urcorner : \mathsf{true}\, \ulcorner P \urcorner\end{array}}{\ulcorner \Delta \urcorner \ \vdash\ \mathsf{forall i}\ \underbrace{(\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)}_{P}\ \underbrace{(\lambda x{:}\mathsf{i}.\, \ulcorner \mathcal{D} \urcorner)}_{D} : \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \underbrace{\ulcorner P \urcorner}_{P\ x}))}$$

- Need to abstract P over x

$$\mathsf{foralli} : \Pi \overbrace{P{:}\mathsf{i} \to \mathsf{o}}^{P}.\ \overbrace{(\Pi x{:}\mathsf{i}.\, \mathsf{true}\ (P\ x))}^{D} \to \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \overbrace{P\ x}^{P\ x}))$$

Representation of Deductions, Universal Elim

- Recall compositionality,
  $\ulcorner [t/x]P \urcorner = [\ulcorner t \urcorner / x]\ulcorner P \urcorner =_\beta (\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)\ \ulcorner t \urcorner$

- Universal elimination

$$\left\ulcorner \dfrac{\begin{array}{c}\mathcal{D} \\ \Delta \vdash \forall x.\, P\ \mathit{true}\end{array} \qquad \begin{array}{c}\mathcal{T} \\ \Delta \vdash t\ \mathit{term}\end{array}}{\Delta \vdash [t/x]P\ \mathit{true}}\,\forall E \right\urcorner$$

$$\dfrac{\begin{array}{rcl} \ulcorner \Delta \urcorner, x{:}\mathsf{i} & \vdash & \ulcorner P \urcorner : \mathsf{o} \\ \ulcorner \Delta \urcorner & \vdash & \ulcorner \mathcal{D} \urcorner : \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \overbrace{\ulcorner P \urcorner}^{P\ x})) \\ \ulcorner \Delta \urcorner & \vdash & \ulcorner t \urcorner : \mathsf{i} \end{array}}{\ulcorner \Delta \urcorner \ \vdash\ \mathsf{foralle}\ \underbrace{(\lambda x{:}\mathsf{i}.\, \ulcorner P \urcorner)}_{P}\ \ulcorner \mathcal{D} \urcorner\ \ulcorner t \urcorner : \mathsf{true}\ \underbrace{([\ulcorner t \urcorner / x]\ulcorner P \urcorner)}_{P\ t}}$$

$$\mathsf{foralle} : \Pi\, \overbrace{P{:}\mathsf{i} \to \mathsf{o}}^{P}\,.\, \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \overbrace{P\ x}^{P\ x})) \to \Pi t{:}\mathsf{i}.\, \mathsf{true}\ \overbrace{(P\ t)}^{P\ t}$$

Representation of Deductions, Summary

- All rules for natural deduction with $\supset$, $\neg$, $\forall$

$$
\begin{array}{rcl}
\mathsf{true} & : & \mathsf{o} \rightarrow \mathit{type} \\
\\
\mathsf{impi} & : & \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, (\mathsf{true}\ P \rightarrow \mathsf{true}\ Q) \rightarrow \mathsf{true}\ (\mathsf{imp}\ P\ Q) \\
\mathsf{impe} & : & \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{imp}\ P\ Q) \ \rightarrow \mathsf{true}\ P \rightarrow \mathsf{true}\ Q \\
\mathsf{noti} & : & \Pi P{:}\mathsf{o}.\, (\Pi q{:}\mathsf{o}.\, \mathsf{true}\ P \rightarrow \mathsf{true}\ q) \rightarrow \mathsf{true}\ (\mathsf{not}\ P) \\
\mathsf{note} & : & \Pi P{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{not}\ P) \rightarrow \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ P \rightarrow \mathsf{true}\ Q \\
\mathsf{foralli} & : & \Pi P{:}\mathsf{i} \rightarrow \mathsf{o}.\, (\Pi x{:}\mathsf{i}.\, \mathsf{true}\ (P\ x)) \rightarrow \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, P\ x)) \\
\mathsf{foralle} & : & \Pi P{:}\mathsf{i} \rightarrow \mathsf{o}.\, \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, P\ x)) \rightarrow \Pi t{:}\mathsf{i}.\, \mathsf{true}\ (P\ t)
\end{array}
$$

- No hidden assumptions or missing definitions!

Adequacy, Revisited

- Representation function is a **compositional bijection** modulo definitional equality in LF
- Proof as before via canonical forms
- Object M represents deduction directly if and only if $\ulcorner \Delta \urcorner \vdash M : \mathsf{true}\, \ulcorner P \urcorner$ and M is canonical
- For an arbitrary object $\ulcorner \Delta \urcorner \vdash N : \mathsf{true}\, \ulcorner P \urcorner$ calculate its unique canonical form
- **Proof checking by type checking in LF**

Representation Example

- Natural deduction

$$\dfrac{\dfrac{\overline{x\ \mathit{term}, u{:}P(x)\ \mathit{true} \vdash P(x)\ \mathit{true}}\ u}{x\ \mathit{term} \vdash P(x) \supset P(x)\ \mathit{true}}\ {\supset}I^u}{\vdash \forall x.\, P(x) \supset P(x)\ \mathit{true}}\ \forall I^x$$

- In LF, for constant or paramater $P{:}\mathsf{i} \to \mathsf{o}$

$$\vdash \mathsf{foralli}\ (\lambda x{:}\mathsf{i}.\, \mathsf{imp}\ (P\ x)\ (P\ x))$$
$$(\lambda x{:}\mathsf{i}.\, \mathsf{impi}\ (P\ x)\ (P\ x)\ (\lambda u{:}\mathsf{true}\ (P\ x).\, u))$$
$$: \mathsf{true}\ (\mathsf{forall}\ (\lambda x{:}\mathsf{i}.\, \mathsf{imp}\ (P\ x)\ (P\ x)))$$

- Note redundant representation of propositions
- Abbreviated form used in practice ([Lect.3] [Necula])

$$\vdash \mathsf{foralli}\ (\lambda x.\, \mathsf{impi}\ (\lambda u.\, u)) : \mathsf{true}\ (\mathsf{forall}\ (\lambda x.\, \mathsf{imp}\ (P\ x)\ (P\ x)))$$

Summary of Lecture 2: Judgments as Types

- Natural deduction (for $\supset$, $\neg$, $\forall$)
- Judgments as types
- Dependent function types in LF
- Hypothetical deductions as functions
- Parametric deduction as dependently typed functions
- Consistent with higher-order abstract syntax
- Renaming of bound variables and substitution immediate
- Representation is compositional bijection
- Proof checking as type checking in LF

Further Examples

- Technique successful in many logics, e.g.,
  - Sequent calculus (2 judgments P *hyp*, P *true*)
  - Hilbert calculus (1 judgment P *valid* [Lect.4])
  - Categorical formulation (1 binary judgment $P \to Q$)
  - Curry-Howard formulation (1 binary judgment $e : P$)
  - Temporal logic (2 judgments P *true at* t, $t \leq t'$)
- Technique successful in programming languages, e.g.,
  - functional programming: typing, evaluation, compilation
  - logic programming: typing, evaluation, compilation
  - more: [notes] [Computation & Deduction, CUP'01]

Limitations of LF

- Limitations are questions of practice, not theory
- Hypotheses not subject to weakening, contraction
- Solution: linear LF based on linear λ-calculus [Cervesato & Pf.'97]
- Hypotheses not subject to exchange
- Solution: ordered LF based on ordered λ-calculus [Polakow'01]
- Built-in theories (integers, reals, strings)
- Approach: LF and dependently typed rewriting, constraints [Necula] [Virga'99]
- Implementation at twelf.org

Preview of Lecture 3: Proof Search and Representation

- Summary of LF
- Canonical forms
- Redundancy elimination
- Constraint logic programming in LF

Logical and Meta-Logical Frameworks
Lecture 3: Proof Search and Representation

- Summary of LF
- Canonical forms
- Redundancy elimination
- Constraint logic programming in LF

Review of Lecture 2: Judgments as Types

- Represent propositions via higher-order abstract syntax
- Represent judgments as types, deductions as objects
- Represent hypothetical deductions as functions
- Represent parametric deductions as dependent functions
- Example: natural deduction
- Representation is compositional bijection
- Inherit renaming and substitution from LF
- Proof checking via type checking in LF

From Simple to Dependent Types

- λ^{Π} type theory from LF generalizes $\lambda^{\rightarrow}$
  - Generalize atomic types a to $a\,M_1 \ldots M_n$, e.g.,
    $\vdash \mathsf{o} : \mathit{type}$ to $q{:}\mathsf{o} \vdash \mathsf{true}\ q : \mathit{type}$
  - Extend type constants a to type families a, e.g.,
    $\vdash \mathsf{o} : \mathit{type}$ to $\vdash \mathsf{true} : \mathsf{o} \rightarrow \mathit{type}$
  - Introduce kinds K and declare $a{:}K$, e.g.,
    $\mathsf{true}{:}\mathsf{o} \rightarrow \mathit{type}$
  - Generalize function types $A \rightarrow B$ to
    dependent function types $\Pi x{:}A.\, B$, e.g.,
    $\mathsf{not} : \mathsf{o} \rightarrow \mathsf{o}$ to
    $\mathsf{note} : \Pi P{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{not}\ P) \rightarrow \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ P \rightarrow \mathsf{true}\ Q$
- $A \rightarrow B = \Pi x{:}A.\, B$ for x not free in B
- $A \rightarrow K = \Pi x{:}A.\, K$ for x not free in K

Example: Classical First-Order Logic

- A rule of classical reasoning

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta, u{:}\neg P\ \mathit{true}, q\ \mathit{prop} \vdash q\ \mathit{true}\end{array}}{\Delta \vdash P\ \mathit{true}}\ \mathsf{contr}$$

- Typing in LF

$$\dfrac{\begin{array}{rcl}\ulcorner\Delta\urcorner & \vdash & \ulcorner P\urcorner : \mathsf{o}\\ \ulcorner\Delta\urcorner, u{:}\mathsf{true}\ (\mathsf{not}\ \ulcorner P\urcorner), q{:}\mathsf{o} & \vdash & \ulcorner\mathcal{D}\urcorner : \mathsf{true}\ q\end{array}}{\begin{array}{rcl}\ulcorner\Delta\urcorner & \vdash & \mathsf{contr}\ \ulcorner P\urcorner\ (\lambda u{:}\mathsf{true}\ (\mathsf{not}\ \ulcorner P\urcorner).\ \lambda q{:}\mathsf{o}.\ \ulcorner\mathcal{D}\urcorner)\\ & & : \mathsf{true}\ \ulcorner P\urcorner\end{array}}$$

- Declaration in LF

$$\mathsf{contr} : \Pi P{:}\mathsf{o}.\ (\mathsf{true}\ (\mathsf{not}\ P) \to \Pi q{:}\mathsf{o}.\ \mathsf{true}\ q) \to \mathsf{true}\ P$$

Summary of LF Type Theory

- Meta-language: λ^{Π} type theory

$$
\begin{array}{rrcl}
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A \\
\textit{Kinds} & K & ::= & \mathit{type} \mid \Pi x{:}A.\, K \\
\textit{Types} & A & ::= & a\, M_1 \, \ldots \, M_n \mid \Pi x{:}A_1.\, A_2 \mid A_1 \to A_2 \\
\textit{Objects} & M & ::= & c \mid x \mid \lambda x{:}A.\, M \mid M_1\, M_2
\end{array}
$$

- Main judgments
  - $\Gamma \vdash_{\Sigma} A : K$ — family A has kind K
  - $\Gamma \vdash_{\Sigma} M : A$ — object M has type A
  - $\Gamma \vdash_{\Sigma} A = B : K$ — A and B are definitionally equal
  - $\Gamma \vdash_{\Sigma} M = N : A$ — M and N are definitionally equal

Critical Rules of LF

- Type conversion (recall: definitial equality is $\beta\eta$)

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B : \mathit{type}}{\Gamma \vdash M : B}\ \mathsf{conv}$$

- Dependent function type, introduction

$$\frac{\Gamma \vdash A : \mathit{type} \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : \Pi x{:}A.\, B}\ \Pi I$$

- Dependent function type, elimination

$$\frac{\Gamma \vdash M : \Pi x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : [N/x]B}\ \Pi E$$

- Dependent kind, elimination

$$\frac{\Gamma \vdash A : \Pi x{:}B.\, K \qquad \Gamma \vdash N : B}{\Gamma \vdash A\ N : [N/x]K}\ \Pi E'$$

Theory of LF

- Complex, because types depend on objects and vice versa
- Complex, because typing depends on equality and vice versa
- Main results [Harper,Honsell,Plotkin'87'93] [Coqand'91] ...
  - Types are unique modulo definitional equality
  - Canonical forms exist and are unique
  - Definitional equality is decidable
  - Type checking is decidable
- New approach to theory [Harper&Pf'00]
- By adequacy: proof checking via LF type checking

Type Checking versus Proof Search

- Type checking (suppressing signature Σ)

  Given Γ, M, A, *decide if* $\Gamma \vdash M : A$

- Type synthesis

  Given Γ, M, *synthesize* A *such that* $\Gamma \vdash M : A$ *or fail*

- Type checking and synthesis are decidable
- Proof search

  Given Γ, A, *search for* M *such that* $\Gamma \vdash M : A$

- Proof search is undecidable

The Central Importance of Canonical Forms

- **Theorem:** For every M such that $\Gamma \vdash M : A$, there is a unique canonical N such that $\Gamma \vdash M = N : A$
- Four applications of canonical forms:
  1. Adequacy theorems formulated on canonical forms

     There is a compositional bijection between deductions $\mathcal{D}$ of $\Delta \vdash P$ true and **canonical** *objects M such that* $\ulcorner\Delta\urcorner \vdash M : \mathsf{true}\,\ulcorner P\urcorner$

  2. Redundancy elimination in representation [Necula]
  3. Focused proof search [Andreoli'91]
  4. Higher-order constraint simplification (unification)
- Caveat: canonical forms may be too large [Statman'78]
- In practice we permit definitions $c : A = M$

Canonical Objects, Definition

- Judgments
  - $\Gamma \vdash M \Downarrow A$ — M is canonical at type A
  - $\Gamma \vdash M \uparrow A$ — M is neutral of type A

- Canonical objects are type-directed

- Canonical objects of function type are λ-abstractions

$$\frac{\Gamma \vdash A \Downarrow \mathit{type} \qquad \Gamma, x{:}A \vdash M \Downarrow B}{\Gamma \vdash \lambda x{:}A.\, M \Downarrow \Pi x{:}A.\, B}\ \Pi I$$

- Canonical objects of atomic type are neutral

$$\frac{\Gamma \vdash M \uparrow a\ M_1 \ldots M_n}{\Gamma \vdash M \Downarrow a\ M_1 \ldots M_n}$$

Neutral Objects, Definition

- Neutral objects are term-directed
- Assume in declarations $c{:}A$ and $x{:}A$, A is canonical
- $can(A)$ calculates canonical form of A
- Variables and constants are neutral

$$\frac{c{:}A \text{ in } \Sigma}{\Gamma \vdash c \uparrow A} \qquad \frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x \uparrow A}$$

- Applications of neutral functions to canonical arguments are neutral

$$\frac{\Gamma \vdash M \uparrow \Pi x{:}A.\, B \qquad \Gamma \vdash N \Downarrow A}{\Gamma \vdash M\ N \uparrow can([N/x]B)}\ \Pi E$$

Application: Bi-Directional Type Checking

- LF so far is based entirely on type synthesis
- Generalize to eliminate all type labels from λ-abstractions without compromising decidability
- Bi-directional checking is robust idea, also applies to
  - subtyping and intersection types [Davies & Pf'00]
  - polymorphic recursion
  - polymorphism and subtyping [Pierce&Turner'00]
- Based on minor variant of canonical forms

Type Checking and Canonical Objects

- Judgments (on objects without type labels)
  - $\Gamma \vdash M \Downarrow A$ — given Γ, M, A, check if $M : A$
  - $\Gamma \vdash M \uparrow A$ — given Γ, M, synthesize A
- Checking at function type ($\Pi x{:}A.\, B$ given)

$$\frac{\Gamma, x{:}A \vdash M \Downarrow B}{\Gamma \vdash \lambda x.\, M \Downarrow \Pi x{:}A.\, B}$$

- Checking at atomic type ($a\ M_1 \ldots M_n$ given)

$$\frac{\Gamma \vdash M \uparrow A \qquad \Gamma \vdash A = a\ M_1 \ldots M_n : \mathit{type}}{\Gamma \vdash M \Downarrow a\ M_1 \ldots M_n}$$

Type Synthesis and Neutral Objects

- Synthesis of variables

$$\frac{c{:}A \text{ in } \Sigma}{\Gamma \vdash c \uparrow A} \qquad \frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x \uparrow A}$$

- Synthesis of applications

$$\frac{\Gamma \vdash M \uparrow \Pi x{:}A.\, B \qquad \Gamma \vdash N \Downarrow A}{\Gamma \vdash M\ N \uparrow [N/x]B}$$

Type Ascription

- No type labels needed for **canonical objects**
- For other objects, introduce type ascription $(M : A)$
- Insert ascription where synthesis is impossible

$$\frac{\Gamma \vdash M \Downarrow A}{\Gamma \vdash (M : A) \Uparrow A}$$

- Example

$$p{:}\mathsf{o} \vdash ((\lambda q.\, q) : \mathsf{o} \to \mathsf{o})\ p \Downarrow \mathsf{o}$$

or (assuming definitions $\textbf{let}\ x{:}A = M\ \textbf{in}\ N$)

$$p{:}\mathsf{o} \vdash \textbf{let}\ q{:}\mathsf{o} = p\ \textbf{in}\ q \Downarrow \mathsf{o}$$

Bi-Directional Checking, Example

- In practice, most objects are canonical
- Example, proof of $\forall x.\, P(x) \supset P(x)$ for parameter $P{:}\mathsf{i} \rightarrow \mathsf{o}$

$$\vdash \mathsf{foralli}\ (\lambda x.\, \mathsf{imp}\ (P\ x)\ (P\ x))\ (\lambda x.\, \mathsf{impi}\ (P\ x)\ (P\ x)\ (\lambda u.\, u))$$
$$\Downarrow \mathsf{true}\ (\mathsf{forall}\ (\lambda x.\, \mathsf{imp}\ (P\ x)\ (P\ x)))$$

- Reduced, but not completely eliminated redundancy

$$\vdash \mathsf{foralli}\ (\lambda x.\, \mathsf{imp}\ (P\ x)\ (P\ x))\ (\lambda x.\, \mathsf{impi}\ (P\ x)\ (P\ x)\ (\lambda u.\, u))$$
$$\Downarrow \mathsf{true}\ (\mathsf{forall}\ (\lambda x.\, \mathsf{imp}\ (P\ x)\ (P\ x)))$$

- Extend the idea of bi-directional checking

Redundant Dependent Arguments

- Recall implication elimination

$$\mathsf{impe} : \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, \mathsf{true}\ (\mathsf{imp}\ P\ Q) \rightarrow \mathsf{true}\ P \rightarrow \mathsf{true}\ Q$$

- Representation (eliding $P{:}\mathsf{o}$ and $Q{:}\mathsf{o}$)

$$\frac{\Gamma \vdash D : \mathsf{true}\ (\mathsf{imp}\ P\ Q) \qquad \Gamma \vdash E : \mathsf{true}\ P}{\Gamma \vdash \mathsf{impe}\ P\ Q\ D\ E : \mathsf{true}\ Q}$$

- Examples of redundancy:
  - If we can synthesize $\Gamma \vdash D \uparrow \mathsf{true}\ (\mathsf{imp}\ P\ Q)$ we can determine P and Q and erase them from $\Gamma \vdash \mathsf{impe}\ P\ Q\ D\ E \uparrow \mathsf{true}\ Q$
  - If we check $\Gamma \vdash \mathsf{impe}\ P\ Q\ D\ E \Downarrow \mathsf{true}\ Q$ we can determine and erase Q but not P

Bi-Directional LF

- Split $\mathsf{true}\ P$ into $\mathsf{true}^{\uparrow}\ P$ and $\mathsf{true}^{\Downarrow}\ P$
- Split each constant into one or several instances
- Either by hand or by LF signature analysis
- $\Gamma \vdash M : \mathsf{true}^{\uparrow}\ P$ must synthesize P
- $\Gamma \vdash M : \mathsf{true}^{\Downarrow}\ P$ checks M against $\mathsf{true}\ P$
- Annotations must be consistent

Bi-Directional LF, Examples

- Analyse types for consistent annotations (by example only)
- !x — we may assume x known
  ?x — we must check if x is known
- Example: implication elimination, standard annotation

$$\mathsf{impe}_1 : \Pi P{:}\mathsf{o}.\,\Pi Q{:}\mathsf{o}.\ \underbrace{\mathsf{true}^{\uparrow}\ P\ Q}_{!P\ !Q} \to \underbrace{\mathsf{true}^{\Downarrow}\ P}_{?P} \to \underbrace{\mathsf{true}^{\uparrow}\ Q}_{?Q}$$

- Example: implication elimination, non-standard annotation

$$\mathsf{impe}_2 : \underbrace{\Pi P{:}\mathsf{o}.}_{!P}\ \Pi Q{:}\mathsf{o}.\ \underbrace{\mathsf{true}^{\Downarrow}\ P\ Q}_{?P\ ?Q} \to \underbrace{\mathsf{true}^{\Downarrow}\ P}_{?P} \to \underbrace{\mathsf{true}^{\Downarrow}\ Q}_{!Q}$$

Bi-Directional LF and Higher-Order Matching

- Example: universal introduction, standard annotation

$$\mathsf{foralli}_1 : \Pi P{:}\mathsf{i} \to \mathsf{o}.\ (\Pi x{:}\mathsf{i}.\ \underbrace{\mathsf{true}^{\Downarrow}\ (P\ x)}_{?P}\ ) \ \to\ \underbrace{\mathsf{true}^{\Downarrow}\ (\mathsf{forall}\ (\lambda x.\, P\ x))}_{!P}$$

- Example: universal elimination, **incorrect** annotation

$$\mathsf{foralle}_1 : \Pi P{:}\mathsf{i} \to \mathsf{o}.\ \underbrace{\mathsf{true}^{\Downarrow}\ (\mathsf{forall}\ (\lambda x.\, P\ x))}_{?P} \to \Pi t{:}\mathsf{i}.\ \underbrace{\mathsf{true}^{\Downarrow}\ (P\ t)}_{!P\ !t}$$

- Problem: even if we know $(P\ t)$ we may not know P and t!

- Example: solve $P\ t = \mathsf{q}\ 0 \supset \mathsf{q}\ 0$ for $P{:}\mathsf{i} \to \mathsf{o}$ and $t{:}\mathsf{i}$:
  $P = (\lambda x.\, \mathsf{q}\ x \supset \mathsf{q}\ x)$ and $t = 0$ or
  $P = (\lambda x.\, \mathsf{q}\ 0 \supset \mathsf{q}\ x)$ and $t = 0$ or
  $P = (\lambda x.\, \mathsf{q}\ 0 \supset \mathsf{q}\ 0)$ and t arbitrary
  etc.

Strict Occurrences

- **Theorem** [Schürmann'00]: Higher-order matching yields a unique answer or fails if every existential variable has **at least one** strict occurrence

- Strict occurrences of P must satisfy two conditions

  1. Have the form $P\, x_1 \ldots x_n$ for distinct parameters x_i
  2. Not be in an argument to an existential variable

- Example: universal elimination with existentials P and t

$$\mathsf{foralle} : \mathsf{true}\ (\mathsf{forall}\ (\lambda x.\ \underbrace{P\ x}_{1})) \rightarrow \mathsf{true}\ (\underbrace{P}_{2}\ \underbrace{t}_{3})$$

1 is strict occurrence of P

2 is not strict (argument t is existential)

3 is not strict (appears in argument to existential P)

Type and Object Reconstruction for LF

- Bi-directional LF requires strict higher-order matching
- Reconstruction is always unique or fails
- For practical experience see [Necula]
- Unrestricted LF requires dependent higher-order unification
- Full reconstruction may have multiple solutions or loop
- Use safe approximation via constraint simplification
- Reconstruction may
  - succeed with principal type
  - fail with error message
  - request more information
- Works well for small objects (see Twelf)

How Do We Compute With Representations?

- LF is functional, but there is no recursion
- Recursion (even prim. rec.) destroys adequacy of encodings
- Counterexample: recall

  $$\mathsf{forall} : (\mathsf{i} \rightarrow \mathsf{o}) \rightarrow \mathsf{o}$$

  Then

  $$\mathsf{forall}\ f : \mathsf{o}$$

  for recursive $f : \mathsf{i} \rightarrow \mathsf{o}$ is not in the image of the $\ulcorner\_\urcorner$
- Also: would violate essential open-endedness
- $\mathsf{i} \rightarrow \mathsf{o}$ must be the parametric function space, i.e., canonical $M : \mathsf{i} \rightarrow \mathsf{o}$ must have the form $\lambda x{:}\mathsf{i}.\ulcorner P \urcorner$ for some P

Constraint Logic Programming with LF

- We cannot easily compute functionally (but [Schürmann,Despeyroux,Pf'97][Schürmann'00])
- Solution: compute as in **constraint logic programming**
- Operational semantics via search with fixed strategy
- Note: **not** general theorem proving
- Related to informal practice of reading rules as algorithms
- Example: bi-directional checking

Example: Recognizing Negation-Free Propositions

- Judgment: $\Delta \vdash P\ \textit{nf}$ supposing $\Delta \vdash P\ \textit{prop}$
- Assume constants $\mathsf{p}{:}\mathsf{i} \to \mathsf{o}$ and $\mathsf{q}{:}\mathsf{o}$
- Four rules:

$$\frac{}{\Delta \vdash q\ \textit{nf}} \qquad \frac{}{\Delta \vdash p\ t\ \textit{nf}}$$

$$\frac{\Delta \vdash P\ \textit{nf} \qquad \Delta \vdash Q\ \textit{nf}}{\Delta \vdash P \supset Q\ \textit{nf}} \qquad \frac{\Delta, x\ \textit{term} \vdash P\ \textit{nf}}{\Delta \vdash \forall x.\, P\ \textit{nf}}$$

- In LF (omitting implicit arguments as in Twelf):

$$\begin{array}{lcl}
\mathsf{nf} & : & \mathsf{o} \to \mathit{type} \\
\\
\mathsf{nfq} & : & \mathsf{nf}\ q \\
\mathsf{nfp} & : & \mathsf{nf}\ (p\ T) \\
\mathsf{nfimp} & : & \mathsf{nf}\ P \to \mathsf{nf}\ Q \to \mathsf{nf}\ (\mathsf{imp}\ P\ Q) \\
\mathsf{nfall} & : & (\Pi x{:}\mathsf{i}.\, \mathsf{nf}\ (P\ x)) \to \mathsf{nf}\ (\mathsf{forall}\ (\lambda x.\, P\ x))
\end{array}$$

Logic Programming Notation in Twelf

- Now reverse the arrows

$$
\begin{array}{lcl}
\mathsf{nf} & : & \mathsf{o} \rightarrow \mathit{type} \\
\\
\mathsf{nfq} & : & \mathsf{nf}\ q \\
\mathsf{nfp} & : & \mathsf{nf}\ (p\ T) \\
\mathsf{nfimp} & : & \mathsf{nf}\ (\mathsf{imp}\ P\ Q) \\
 & & \quad \leftarrow \mathsf{nf}\ Q \\
 & & \quad \leftarrow \mathsf{nf}\ P \\
\mathsf{nfall} & : & \mathsf{nf}\ (\mathsf{forall}\ (\lambda x.\, P\ x)) \\
 & & \quad \leftarrow (\Pi x{:}\mathsf{i}.\, \mathsf{nf}\ (P\ x))
\end{array}
$$

- Given a query $\mathsf{nf}\ \mathbf{P}$ for a closed, ground $\mathbf{P}$
  match heads of rules in order,
  then solve subgoals in order

A Program Elimination Double Negation

```
q : o.
p : i -> o.

nf : o -> type.
%mode nf +P.

nfq : nf q.
nfp : nf (p T).
nfimp : nf (P imp Q)
         <- nf P
         <- nf Q.
nfall : nf (forall [x] P x)
         <- ({x:i} nf (P x)).

%query 1 * nf (forall [x] p x imp p x).
%query 0 * nf (forall [x] not (p x)).
```

Constraint Simplification in Twelf

- Given example requires only strict higher-order matching (goal has no existential variables, heads are strict)
- In general requires higher-order unification (non-deterministic and undecidable)
- Implemented instead as constraint simplification (pattern unification [Miller'91] + constraints [Pf'91'96])
- Success with constraints is conditional: Any solution to remaining constraints is solution to query
- Methodology: write programs to lie within the strict higher-order matching fragment whenever possible

Operational Semantics of Twelf as in Prolog

- Solve subgoal $\Pi x{:}A.\, B$ by assuming $x{:}A$ and solving B
- When goal is atomic, unify with head of each hypothesis and constant in order
- When heads unify, solve subgoals from left to right
- Backtrack upon failure to most recent choice point
- In general only non-deterministically complete:
  - Finite failure implies no deduction can exist
  - May loop on judgment with a deduction
- Technique: focused proofs [Andreoli'90], uniform proofs [Miller,Nadathur,Pf.,Scredov'91]

Experience with Logic Programming in Twelf

- Many algorithms can be specified at a very high level
- A few algorithms can be very difficult
  (e.g., non-parametric operations)
- Not intended for general purpose programming,
  (e.g., no cut, input/output, other impure features)
- Often possible to prove correctness inside Twelf [Lect.4]
- Examples:
  cut-elimination, logical interpretations, type checking, type inference, evaluation, compilation

Another Example: Eliminating Double Negations

- elim $\mathbf{P}$ Q with input $\mathbf{P}$ generates output Q
- This "directionality" is called a mode
- Can be checked in Twelf implementation

Program in Twelf

```
elim : o -> o -> type.
%mode elim +P -Q.

eq : elim q q.
ep : elim (p T) (p T).
eimp : elim (P1 imp P2) (Q1 imp Q2)
        <- elim P1 Q1
        <- elim P2 Q2.
eall : elim (forall [x] P x) (forall [x] Q x)
        <- ({x:i} elim (P x) (Q x)).
enn : elim (not (not P)) Q
       <- elim P Q.
enq : elim (not q) (not q).
enp : elim (not (p T)) (not (p T)).
enimp : elim (not (P1 imp P2)) (not (Q1 imp Q2))
         <- elim P1 Q1
         <- elim P2 Q2.
enall : elim (not (forall [x] P x)) (not (forall [x] Q x))
         <- ({x:i} elim (P x) (Q x)).
```

A Query and Answer in Twelf

```
%query 1 *
M : elim (not (not q) imp forall [x] p x imp p x) Q.

---------- Solution 1 ----------
Q = q imp forall ([x:i] p x imp p x).
M = eimp (eall ([x:i] eimp ep ep)) (enn eq).
____________________________________________
```

Summary of Lecture 3: Proof Search and Representation

- LF type theory is dependently typed λ-calculus
- Absence of recursion is crucial for adequacy
- Existence and uniqueness of canonical forms is crucial:
  - adequacy theorems
  - redundancy elimination in representation [Necula]
  - strict higher-order matching and constraint simplification
  - focused and uniform proof search
- Implementing algorithms via constraint logic programming
- Specifications and implementations in the same language!

Preview of Lecture 4: Meta-Logical Frameworks

- Hilbert's axiomatic calculus in LF
- The Deduction Theorem
- Meta-theoretic proofs as judgments relating derivations
- Mode, termination, and coverage checking for verification
- Summary

Logical and Meta-Logical Frameworks
Lecture 4: Meta-Logical Frameworks

- Hilbert's axiomatic calculus in LF
- The Deduction Theorem
- Meta-theoretic proofs as judgments relating dedeductions
- Mode, termination, and coverage checking for verification
- Summary
- **Note:** in this lecture, "proof" always refers to meta-theory of deductive systems (encoded in LF)

Review of Lecture 3: Proof Search and Representation

- Central role of canonical forms:
  - adequacy theorems
  - bi-directional type-checking and redundancy elimination
  - strict higher-order matching and constraint simplification
  - focused and uniform proof search
- Absence of recursion is crucial
- Implementing algorithms via constraint logic programming
- Specifications and implementations in the same language!

Hilbert's Axiomatic Calculus

- Judgment $\Delta \vdash P$ *valid* for $\Delta \vdash P\ prop$
- $\Delta = x_1$ *term*, $\ldots, x_n$ *term* (no assumptions Q *true* or Q *valid*)
- Many axioms (= inference rules with no premises)

K $\quad \Delta \vdash P \supset (Q \supset P)$ *valid*

S $\quad \Delta \vdash (P \supset (Q \supset R)) \supset (P \supset Q) \supset (P \supset R)$ *valid*

N_1 $\quad \Delta \vdash (P \supset \neg Q) \supset ((P \supset Q) \supset \neg P)$ *valid*

N_2 $\quad \Delta \vdash \neg P \supset (P \supset Q)$ *valid*

F_1 $\quad \Delta \vdash (\forall x.\, P) \supset [t/x]P$ *valid*

F_2 $\quad \Delta \vdash (\forall x.\, Q \supset P) \supset (Q \supset \forall x.\, P)$ *valid* $\quad$ (x not free in Q)

Two Inference Rules

- Modus Ponens

$$\frac{\Delta \vdash P \supset Q \textit{ valid} \qquad \Delta \vdash P \textit{ valid}}{\Delta \vdash Q \textit{ valid}} MP$$

- Universal Generalization

$$\frac{\Delta, x \textit{ term} \vdash P \textit{ valid}}{\Delta \vdash \forall x.\, P \textit{ valid}} UG^x$$

Representation in Twelf

```
valid : o -> type.

k : valid (P imp (Q imp P)).
s : valid ((P imp (Q imp R)) imp ((P imp Q) imp (P imp R))).

n1 : valid ((P imp (not Q)) imp ((P imp Q) imp (not P))).
n2 : valid ((not P) imp (P imp Q)).

f1 : {T:i} valid ((forall [x:i] P x) imp (P T)).
f2 : valid ((forall [x:i] (Q imp P x))     % incorporates proviso!
             imp (Q imp forall [x:i] P x)).

mp : valid (P imp Q) -> valid P -> valid Q.
ug : ({x:i} valid (P x)) -> valid (forall [x:i] P x).
```

The Deduction Theorem

- **Theorem:** If $\Delta, P\ \mathit{valid} \vdash Q\ \mathit{valid}$ then $\Delta \vdash (P \supset Q)\ \mathit{valid}$
- **Proof:** By induction on the deduction $\mathcal{H}$ of $\Delta, P\ \mathit{valid} \vdash Q\ \mathit{valid}$.
- **Case:** $\mathcal{H}$ ends in the hypothesis rule

$$\frac{}{\Delta, P\ \mathit{valid} \vdash P\ \mathit{valid}}\ \mathsf{hyp}$$

Then (written in abbreviated form)

1	$(P \supset ((P \supset P) \supset P)) \supset ((P \supset (P \supset P)) \supset (P \supset P))$	S
2	$(P \supset ((P \supset P) \supset P))$	K
3	$(P \supset (P \supset P)) \supset (P \supset P)$	MP 1 2
4	$P \supset (P \supset P)$	K
5	$P \supset P$	MP 3 4

Axiom Cases

- **Case:** $\mathcal{H}$ ends in axiom K

$$\frac{}{\Delta, P \; valid \vdash (Q_1 \supset (Q_2 \supset Q_1)) \; valid} K$$

Then

1	$(Q_1 \supset (Q_2 \supset Q_1)) \supset (P \supset (Q_1 \supset (Q_2 \supset Q_1)))$	K
2	$Q_1 \supset (Q_2 \supset Q_1)$	K
3	$P \supset (Q_1 \supset (Q_2 \supset Q_1))$	$MP\,1\,2$

- Other axiom cases analogous

Modus Ponens

- **Case:** $\mathcal{H}$ ends in Modus Ponens

$$\mathcal{H} = \dfrac{\begin{array}{cc} \mathcal{H}_1 & \mathcal{H}_2 \\ \Delta, P\,\mathit{valid} \vdash Q_1 \supset Q_2\ \mathit{valid} & \Delta, P\,\mathit{valid} \vdash Q_1\ \mathit{valid} \end{array}}{\Delta, P\,\mathit{valid} \vdash Q_2\ \mathit{valid}}\ MP$$

1	$\Delta \vdash P \supset (Q_1 \supset Q_2)\ \mathit{valid}$	IH on $\mathcal{H}_1$
2	$\Delta \vdash (P \supset (Q_1 \supset Q_2)) \supset ((P \supset Q_1) \supset (P \supset Q_2))\ \mathit{valid}$	S
3	$\Delta \vdash (P \supset Q_1) \supset (P \supset Q_2)\ \mathit{valid}$	MP 2 1
4	$\Delta \vdash P \supset Q_1\ \mathit{valid}$	IH on $\mathcal{H}_2$
5	$\Delta \vdash P \supset Q_2\ \mathit{valid}$	MP 3 4

Universal Generalization

- **Case:** $\mathcal{H}$ ends in Universal Generalization:

$$\mathcal{H} = \cfrac{\begin{array}{c}\mathcal{H}_1\\ \Delta, x\ \textit{term}, P\ \textit{valid} \vdash Q_1\ \textit{valid}\end{array}}{\Delta, P\ \textit{true} \vdash \forall x.\, Q_1\ \textit{valid}}\ UG^x$$

1	$\Delta, x\ \textit{term} \vdash P \supset Q_1\ \textit{valid}$	IH. on $\mathcal{H}_1$
2	$\Delta \vdash \forall x.\,(P \supset Q_1)\ \textit{valid}$	UG^x 1
3	$\Delta \vdash (\forall x.\,(P \supset Q_1)) \supset (P \supset \forall x.\, Q_1)\ \textit{valid}$	F_2
4	$\Delta \vdash P \supset \forall x.\, Q_1\ \textit{valid}$	MP 3 2

- QED

A Task for a Meta-Logical Framework

- How do we represent this proof?
- Simpler question: what is its computational contents?
- Answer: a translation of deductions $\Delta, P\ \mathit{valid} \vdash Q\ \mathit{valid}$ to deductions of $\Delta \vdash (P \supset Q)\ \mathit{valid}$
- Or, after representation (ignoring Δ):

$$\mathsf{ded} : \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, (\mathsf{valid}\ P \to \mathsf{valid}\ Q) \to \mathsf{valid}\ (\mathsf{imp}\ P\ Q)$$

- This function would be defined by recursion (induction) over

$$H : (\mathsf{valid}\ P \to \mathsf{valid}\ Q)$$

- What does this mean?
- Anyway, recursive functions cannot be part of LF

Possible Answers

- Give up on higher-order abstract syntax and use inductive encodings [many refs]
  - Lose advantages of renaming and substitution!
  - More indirect encodings and more difficult formal proofs
- Use same trick as for algorithms! [Pf'89'91]
  - Implement computational contents of proof as a **logic program**
  - Verify that this logic program describes a proof
  - "*Logic programs as realizers*"
- Other approaches [Despeyroux et al.'94'98] [McDowell&Miller'97] [Schürmann&Pf'98] [Hofmann'99] [Gabbay&Pitts'99] [Schürmann'00'01]

Proofs as Relations

- The proof of the deduction theorem describes a judgment relating deductions of $\Delta, P\ \textit{valid} \vdash Q\ \textit{valid}$ and $\Delta \vdash (P \supset Q)\ \textit{valid}$
- In LF:

$$\mathsf{ded} : \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, (\mathsf{valid}\ P \to \mathsf{valid}\ Q) \to \mathsf{valid}\ (\mathsf{imp}\ P\ Q) \to \textit{type}$$

- This can be represented easily, case by case
- Elide P and Q as in implementation

Hypothesis Case

- **Case:** $\mathcal{H}$ ends in the hypothesis rule

$$\frac{}{\Delta, P\ \mathit{valid} \vdash P\ \mathit{valid}}\ \mathsf{hyp}$$

Then (written in abbreviated form)

1	$(P \supset ((P \supset P) \supset P)) \supset ((P \supset (P \supset P)) \supset (P \supset P))$	S
2	$(P \supset ((P \supset P) \supset P))$	K
3	$(P \supset (P \supset P)) \supset (P \supset P)$	MP 1 2
4	$P \supset (P \supset P)$	K
5	$P \supset P$	MP 3 4

- Recall ded : (valid $P \rightarrow$ valid Q) $\rightarrow$ valid (imp P Q) $\rightarrow$ *type*
- This case ded_id : ded $(\lambda u.\, u)$ (mp (mp s k) k)

Axiom Cases

- **Case:** $\mathcal{H}$ ends in axiom K

$$\frac{}{\Delta, P\ \mathit{valid} \vdash (Q_1 \supset (Q_2 \supset Q_1))\ \mathit{valid}}\ K$$

  Then

1	$(Q_1 \supset (Q_2 \supset Q_1)) \supset (P \supset (Q_1 \supset (Q_2 \supset Q_1)))$	K
2	$Q_1 \supset (Q_2 \supset Q_1)$	K
3	$P \supset (Q_1 \supset (Q_2 \supset Q_1))$	$MP\,1\,2$

- Recall $\mathsf{ded} : (\mathsf{valid}\ P \to \mathsf{valid}\ Q) \to \mathsf{valid}\ (\mathsf{imp}\ P\ Q) \to \mathit{type}$

- This case:

$$\mathsf{ded\_k} : \mathsf{ded}\ (\lambda u.\, \mathsf{k})\ (\mathsf{mp}\ \mathsf{k}\ \mathsf{k})$$

- Other axiom cases are analogous

Modus Ponens

- **Case:** $\mathcal{H}$ ends in Modus Ponens

$$\mathcal{H} = \dfrac{\begin{array}{cc} \mathcal{H}_1 & \mathcal{H}_2 \\ \Delta, P\ \mathit{valid} \vdash Q_1 \supset Q_2\ \mathit{valid} & \Delta, P\ \mathit{valid} \vdash Q_1\ \mathit{valid} \end{array}}{\Delta, P\ \mathit{valid} \vdash Q_2\ \mathit{valid}} MP$$

1	$\Delta \vdash P \supset (Q_1 \supset Q_2)\ \mathit{valid}$	IH on $\mathcal{H}_1$
2	$\Delta \vdash (P \supset (Q_1 \supset Q_2))$	
	$\supset ((P \supset Q_1) \supset (P \supset Q_2))\ \mathit{valid}$	S
3	$\Delta \vdash (P \supset Q_1) \supset (P \supset Q_2)\ \mathit{valid}$	MP 2 1
4	$\Delta \vdash P \supset Q_1\ \mathit{valid}$	IH on $\mathcal{H}_2$
5	$\Delta \vdash P \supset Q_2\ \mathit{valid}$	MP 3 4

- Appeal to induction hypothesis as recursive call

$$\begin{array}{lll} \mathsf{ded\_mp} & : & \mathsf{ded}\ (\lambda u.\ \mathsf{mp}\ (H_1\ u)\ (H_2\ u))\ (\mathsf{mp}\ (\mathsf{mp}\ s\ H_1')\ H_2') \\ & & \quad \leftarrow \mathsf{ded}\ (\lambda u.\ H_1\ u)\ H_1' \\ & & \quad \leftarrow \mathsf{ded}\ (\lambda u.\ H_2\ u)\ H_2' \end{array}$$

Universal Generalization

- **Case:** $\mathcal{H}$ ends in Universal Generalization:

$$\mathcal{H} = \dfrac{\begin{array}{c}\mathcal{H}_1\\ \Delta, x\ \mathit{term}, P\ \mathit{valid} \vdash Q_1\ \mathit{valid}\end{array}}{\Delta, P\ \mathit{true} \vdash \forall x.\, Q_1\ \mathit{valid}}\ UG^x$$

1	$\Delta, x\ \mathit{term} \vdash P \supset Q_1$	IH. on $\mathcal{H}_1$
2	$\Delta \vdash \forall x.\, (P \supset Q_1)$	UG^x 1
3	$\Delta \vdash (\forall x.\, (P \supset Q_1)) \supset (P \supset \forall x.\, Q_1)$	F_2
4	$\Delta \vdash P \supset \forall x.\, Q_1$	MP 3 2

- Appeal to induction hypothesis as recursive call

$$\begin{array}{rcl}\mathsf{ded\_ug} & : & \mathsf{ded}\ (\lambda u.\, \mathsf{ug}\ (\lambda x.\, H_1\ u\ x))\ (\mathsf{mp}\ \mathsf{f2}\ (\mathsf{ug}\ H_1'))\\ & & \leftarrow \Pi x{:}\mathsf{i}.\, \mathsf{ded}\ (\lambda u.\, H_1\ u\ x)\ (H_1'\ x)\end{array}$$

- QED

Executing the Proof Representation

- One can now execute the proof as a logic program with queries

$$\text{ded } \mathbf{H} \; H'$$

  where $\mathbf{H}$ is a given hypothetical deduction and H' is a variable that will be bound to the output deduction

- Computational content fully represented

- We know each output will be correct by adequacy

$$\text{ded} : (\text{valid } P \to \text{valid } Q) \to \text{valid } (\text{imp } P \; Q) \to type$$

Is the Program a Proof?

- Just knowing

  $$\mathsf{ded} : \Pi P{:}\mathsf{o}.\, \Pi Q{:}\mathsf{o}.\, (\mathsf{valid}\ P \to \mathsf{valid}\ Q) \to \mathsf{valid}\ (\mathsf{imp}\ P\ Q) \to \mathit{type}$$

  is not enough

- Need

  For every $\Delta = x_1{:}\mathsf{i}, \ldots, x_n{:}\mathsf{i}$
  and every object P such that $\Delta \vdash P : \mathsf{o}$
  and every object Q such that $\Delta \vdash Q : \mathsf{o}$
  and every object H such that $\Delta \vdash H : (\mathsf{valid}\ P \to \mathsf{valid}\ Q)$
  there exists an H' such that $\Delta \vdash H' : \mathsf{valid}\ (\mathsf{imp}\ P\ Q)$
  and an M such that $\Delta \vdash M : \mathsf{ded}\ P\ Q\ H\ H'$

Proof Verification

- How could this property fail for a type-correct query?

$$\mathsf{ded}\ \mathbf{H}\ H'$$

  - H' could fail to be ground — mode checking
  - Query could fail to terminate — termination checking
  - Query could fail finitely — coverage checking

- Mode, termination, and coverage checking together with adequacy of representation guarantee that the type family ded implements a proof of the deduction theorem

Mode Checking

- Quite straightforward, using strictness

```
ded : (valid P -> valid Q) -> valid (P imp Q) -> type.
%mode ded +H -H'.

ded_mp : ded ([u] mp (H1 u) (H2 u)) (mp (mp s H1') H2')
          <- ded ([u] H1 u) H1'
          <- ded ([u] H2 u) H2'.
```

- Input argument (+):
  assume ground for head, check ground for recursive call
- Output argument (−):
  assume ground for recursive call, check ground for head
- Good, informative error messages!

Termination Checking

- Assume user gives termination order
- Based on subterm ordering corresponding to structural induction

```
ded : (valid P -> valid Q) -> valid (P imp Q) -> type.
%terminates H (ded H _)

ded_mp : ded ([u] mp (H1 u) (H2 u)) (mp (mp s H1') H2')
          <- ded ([u] H1 u) H1'
          <- ded ([u] H2 u) H2'.
```

Termination Checking in Twelf

- Can construct lexicographic and simultaneous orders
- Difficult part: higher-order subterm orderings [Pientka]
- Explicit specification expresses "*By induction over* $\mathcal{H}$"
- Informative error messages
- Improve checking mutual recursion [Abel][Jones]

Coverage Checking

- Guarantees that for every combination of (ground) inputs some clause applies
- Coverage entails progress (no finite failure)
- Difficult, because it contradicts open-endedness
- Inherently, to check an inductive proof, we need to fix the set of constructors
- No paradoxes, since there is no new object constructor

Regular Worlds

- Recall

  For every $\Delta = x_1{:}\mathsf{i}, \ldots, x_n{:}\mathsf{i}$
  and every object P such that $\Delta \vdash P : \mathsf{o}$
  and every object Q such that $\Delta \vdash Q : \mathsf{o}$
  and every object H such that $\Delta \vdash H : (\mathsf{valid}\ P \to \mathsf{valid}\ Q)$
  there exists an H' such that $\Delta \vdash H' : \mathsf{valid}\ (\mathsf{imp}\ P\ Q)$
  and an M such that $\Delta \vdash M : \mathsf{ded}\ P\ Q\ H\ H'$

- Need to describe the form of possible contexts

- Use regular worlds defined schematically [Schürmann00]

$$\Delta_{\mathsf{ded}} ::= \cdot \mid \Delta_{\mathsf{ded}}, x{:}\mathsf{i}$$

Coverage Checking

- With respect to regular world definition (e.g., Δ_{ded})
- Coverage set = exhaustive set of possible query shapes
- Initialize with most general query $\mathsf{ded}\ H\ \_$
- Algorithm:
  1. Pick and remove a query shape G from the coverage set
  2. Check if G is an instance of a clause head (strict higher-order matching)
  3. If not, pick a candidate variable (halt if none), generate all possible instances (higher-order unification) and add them to the coverage set
  4. Go to 1.
- Re-implementation still in progress (not available in current Twelf)

Implementing Meta-Theoretic Proofs, Summary

- Represent computational contents as judgment relating deductions
  (here: ded : (valid $P \to$ valid Q) $\to$ valid (imp P Q) $\to$ *type*)
- Together
  - dependent type checking (no invalid deductions)
  - mode checking (no missing constructors)
  - termination checking (no divergence)
  - coverage checking (no finite failure)

  guarantee that implementation represents meta-theoretic proof
- All of these are efficiently decidable with good or acceptable error messages
- **Logic Programs as Proofs**

Experience with Relational Meta-Theory

- Proofs are often very compact
  - Immediacy of encoding (hoas, judgments as types)
  - Type reconstruction
- Applicable in many case studies
  - logical interpretations (nd vs axiomatic, nd vs sequent, classical vs intuitionistic, nd vs categorical)
  - logical properties (cut elimination, normalization, deduction theorem)
  - λ-calculus (CR theorem, CPS transform)
  - small programming languages (functional, logic) (type preservation and progress for various type systems, compiler correctness)
- Used succesfully in teaching several times

Automation

- Due to high level of representation, many meta-theorems can be proven **automatically** [Schürmann&Pf'98] [Schürmann'00]
- Input: specification, $\forall\exists$ meta-theorem, induction order
- Output: proof in relational form
- Alternate direct search in LF (bounded depth-first search) with case splitting
- Often very fast (type preservation, deduction theorem)
- Not very robust with respect to signature extension
- Not very robust with respect to number of inputs

Some Limitations

- Logical relations or reducibility candidates [Girard'71]
- Where encodings are awkward (linear, ordered), proofs are infeasible
- Proofs are "write only"
- Some work on "uncompressing" into readable format (TCS paper on cut elimination 50% written by machine)

Summary

- Meta-logical frameworks for reasoning about deductive systems
- Two choices
  - Techniques for representation:
    usually inductive (low level), here judgments as types
  - Techniques for proof representation:
    usually recursive functions, here judgments relating derivations
  - Techniques for proof checking:
    similar in both approaches
- Various hybrid techniques have been investigated
- High-level representation facilitates both manual and automatic proofs

Course Summary

- **Lecture 1**: Higher-Order Abstract Syntax
  Variables as variables, representation is compositional bijection, substitution as substitution
- **Lecture 2**: Judgments as Types
  Parametric judgments as functions, checking deductions via type checking in LF
- **Lecture 3**: Search and Representation
  Canonical forms, bi-directional checking, logic programming
- **Lecture 4**: Meta-Logical Frameworks
  Meta-theoretic proofs as judgments relating derivations, checking modes, termination, coverage

Course Slogans

- **Specifications, algorithms, meta-theory in the same minimal language** (only type constructor: $\Pi x{:}A.\, B$!)
- **Elegance matters!**
- We had to slaughter some holy cows:
  - inductive types and explicit induction principles
  - tactic-based theorem proving
- Logical frameworks are **not** for general mathematics

On the Horizon

- Module system
- Constraint domains (rationals)
- Linearity and order in the framework
- Compression of deductions
- Specialization with respect to fixed signature?

Reference Material

- Lecture Material:

  Logical frameworks.
  Handbook of Automated Reasoning,
  Chapter 16, pp. 977-1061,
  Elsevier Science and MIT Press, June 2001.

- Textbook:

  Computation and Deduction.
  Cambridge University Press, Fall 2001.

- Implementation: twelf.org