Curry-Howard Correspondence for Intuitionistic Propositional Logic

Lecture Interactive Proof Tools

Summer Term 2003

Hans de Nivelle, Patrick Maier

Contents

- Untyped λ-calculus.
- Simply typed λ-calculus.
- Minimal logic (implicational fragment of intutionsitic propositional logic).
- Curry-Howard correspondence for minimal logic.

Untyped λ-calculus – Pre-terms

Definitions:

- Assume an infinite alphabet of *variables* V, ranged over by u, v, x, y, z.
- *Pre-terms* Λ^- are defined by the grammar

$$
\begin{array}{rcll}
\Lambda^- & ::= & V & \textit{variable} \\
 & \mid & (\Lambda^- \, \Lambda^-) & \textit{application} \\
 & \mid & (\lambda V \Lambda^-) & \textit{abstraction}
\end{array}
$$

  and ranged over by K, L, M, N.

Untyped λ-calculus – Notation

Conventions to save parentheses:

- Omit outermost parentheses.
- Nested application: Write KLM for $(KL)M$.
- Nested abstraction: Write $\lambda x \lambda y M$ for $\lambda x(\lambda y M)$.
- Application under abstraction: Write $\lambda x M N$ for $\lambda x(MN)$.
- Abstraction under application: Write $M \lambda x N$ for $M(\lambda x N)$.

Convention to save lambdas:

- n-fold abstraction: Write $\lambda x_1 \ldots x_n.M$ for $\lambda x_1 \ldots \lambda x_n M$.

Untyped λ-calculus – Free Variables

Define the *free variables* $\mathrm{FV}(M)$ of M inductively over M by

$$
\begin{array}{rcl}
\mathrm{FV}(x) & = & \{x\} \\
\mathrm{FV}(KL) & = & \mathrm{FV}(K) \cup \mathrm{FV}(L) \\
\mathrm{FV}(\lambda x K) & = & \mathrm{FV}(K) \setminus \{x\}
\end{array}
$$

We say that M is *closed* iff $\mathrm{FV}(M) = \emptyset$.

Untyped λ-calculus – Substitutions

Define $M[x := N]$ inductively over M (assuming $x \neq y$) by

$$
\begin{array}{rcl}
x[x := N] & = & N \\
y[x := N] & = & y \\
(K\,L)[x := N] & = & (K[x := N])(L[x := N]) \\
(\lambda x K)[x := N] & = & \lambda x K \\
(\lambda y K)[x := N] & = & \lambda z((K[y := z])[x := N]) \\
 & & \text{where } z \textit{ fresh}, \text{ i. e., } z \notin \mathrm{FV}(K) \cup \mathrm{FV}(N) \cup \{x\}
\end{array}
$$

Substitution binds tighter than application and abstraction:

- Write $K[x := N]\,L[x := N]$ for $(K[x := N])(L[x := N])$.
- Write $\lambda z K[y := z][x := N]$ for $\lambda z((K[y := z])[x := N])$.

Untyped λ-calculus – α-equivalence

Define $\equiv_\alpha$ as the smallest equivalence relation on Λ^- closed under

$$\begin{array}{rcll}
y \notin \mathrm{FV}(M) & \Rightarrow & \lambda x M \equiv_\alpha \lambda y M[x := y] & \text{renaming of bound variables,} \\
M \equiv_\alpha N & \Rightarrow & \lambda x M \equiv_\alpha \lambda x N & \text{abstraction,} \\
M \equiv_\alpha N & \Rightarrow & K M \equiv_\alpha K N & \text{application (left)} \\
M \equiv_\alpha N & \Rightarrow & M K \equiv_\alpha N K & \text{application (right).}
\end{array}$$

Properties of $\equiv_\alpha$:

- If $K \equiv_\alpha L$ then $\mathrm{FV}(K) = \mathrm{FV}(L)$.
- If $x \notin \mathrm{FV}(K)$ then $K[x := N] \equiv_\alpha K$.
- $K \equiv_\alpha L$ iff $K[x := N] \equiv_\alpha L[x := N]$.

Untyped λ-calculus – Terms

Define α-equivalence classes $[M]_\alpha$ and the set of *terms* Λ by

$$[M]_\alpha = \{N \in \Lambda^- \mid M \equiv_\alpha N\}$$

$$\Lambda = \Lambda^- / \equiv_\alpha = \{[M]_\alpha \mid M \in \Lambda^-\}$$

Convention: We identify terms with pre-terms i. e., we denote the term $[M]_\alpha$ by the pre-term M, and we assume that all operations defined on terms are compatible with α-equivalence.

Exercise: Convince yourself that free variables and substitution are compatible with $\equiv_\alpha$:

- If $K \equiv_\alpha L$ then $\mathrm{FV}(K) = \mathrm{FV}(L)$.
- If $K \equiv_\alpha L$ and $M \equiv_\alpha N$ then $K[x := M] \equiv_\alpha L[x := N]$.

Untyped λ-calculus – Sample Terms

Some important closed terms:

- combinator I, identity function: $\lambda x.x$
- combinator K, first projection, constant function: $\lambda xy.x$
- combinator K*, second projection: $\lambda xy.y$
- combinator S: $\lambda xyz.xz(yz)$
- term ω, self-application: $\lambda x.xx$
- term Ω, self-self-application: $(\lambda x.xx)\lambda x.xx$

Note that $\lambda xy.x = \lambda yx.y = \lambda zy.z$ since we identify α-equivalent terms.

Untyped λ-calculus – β-reduction

Definitions:

- Define $\rightarrow_\beta$ as the smallest binary relation on Λ closed under

$$
\begin{array}{rcll}
 & (\lambda x M)N \rightarrow_\beta M[x := N] & & \beta\text{-contraction,} \\
M \rightarrow_\beta N & \Rightarrow & \lambda x M \rightarrow_\beta \lambda x N & \text{abstraction,} \\
M \rightarrow_\beta N & \Rightarrow & KM \rightarrow_\beta KN & \text{application (left)} \\
M \rightarrow_\beta N & \Rightarrow & MK \rightarrow_\beta NK & \text{application (right).}
\end{array}
$$

- $(\lambda x M)N$ is a *β-redex* and $M[x := N]$ is a *β-contractum.*
- M is in *β-normal form* iff there is no N with $M \rightarrow_\beta N$.
- Define $\rightarrow_\beta^*$ as the reflexive-transitive closure of $\rightarrow_\beta$.
- Define $\leftrightarrow_\beta^*$ as the reflexive-transitive-symmetric closure of $\rightarrow_\beta$.

Untyped λ-calculus – Sample β-reductions

- $(\lambda x.x)y \rightarrow_\beta x[x := y] = y$.
- $(\lambda xy.x)yz = ((\lambda x.(\lambda y.x))y)z \rightarrow_\beta (\lambda y.x[x := y])z$
$$= (\lambda ux[y := u][x := y])z = (\lambda ux[x := y])z$$
$$= (\lambda uy)z \rightarrow_\beta y[u := z] = y.$$
- $(\lambda x.xx)yy \rightarrow_\beta (xx[x := y])y = (yy)y$.
- $(\lambda x.xx)(yy) \rightarrow_\beta xx[x := (yy)] = (yy)(yy)$.
- $(\lambda x.xx)(\lambda x.xx) \rightarrow_\beta xx[x := (\lambda x.xx)] = (\lambda x.xx)(\lambda x.xx)$.

Untyped λ-calculus – Properties of $\rightarrow_\beta$

- Compatible with $\equiv_\alpha$:
  If $K \equiv_\alpha M$ and $L \equiv_\alpha N$ and $K \rightarrow_\beta L$ then $M \rightarrow_\beta N$.
- Church-Rosser property:
  If $K \rightarrow_\beta^* L_1$ and $K \rightarrow_\beta^* L_2$ then there is M such that $L_1 \rightarrow_\beta^* M$ and $L_2 \rightarrow_\beta^* M$.
- Corollary: If $L_1 \leftrightarrow_\beta^* L_2$ then there is M such that $L_1 \rightarrow_\beta^* M$ and $L_2 \rightarrow_\beta^* M$.
- Corollary: If L_1, L_2 in β-normal from then $L_1 \leftrightarrow_\beta^* L_2$ iff $L_1 = L_2$.
- All recursive functions are λ-definable.
- Corollary: $\leftrightarrow_\beta^*$ is undecidable.

Simply Typed λ-calculus – Why

- Untyped λ-calculus can express too many functions
  $\rightsquigarrow$ undecidability of β-equivalence.

- For β-normal forms, β-equivalence is trivial
  $\rightsquigarrow$ if β-normalforms exist then reduce deciding β-equivalence to normalization (computation of β-normal forms).

- Type system restricts λ-terms such that β-normal forms exist.

Simply Typed λ-calculus – Types

Definitions:

- Assume an infinite alphabet of *type variables* U, ranged over by α, β, γ.
- *Simple types* Π are defined by the grammar

$$
\begin{array}{rcll}
\Pi & ::= & U & \textit{type variable} \\
& | & (\Pi \rightarrow \Pi) & \textit{function type}
\end{array}
$$

  and ranged over by σ, τ, ρ.

Notation:

- Omit outermost parentheses.
- Write $\sigma \rightarrow \tau \rightarrow \rho$ for $\sigma \rightarrow (\tau \rightarrow \rho)$.

Simply Typed λ-calculus – Contexts

Definitions:

- A *context* $\Gamma \in C$ is a set of the form $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ where each x_i occurs only once.
- Define the *domain* $\mathrm{dom}(\Gamma)$ of Γ by

$$\mathrm{dom}(\{x_1 : \tau_1, \ldots, x_n : \tau_n\}) = \{x_1, \ldots, x_n\}.$$

- Define the *range* $|\Gamma|$ of Γ by

$$|\{x_1 : \tau_1, \ldots, x_n : \tau_n\}| = \{\tau_1, \ldots, \tau_n\}.$$

Notation:

- Omit set braces and write contexts as comma-separated lists.

Simply Typed λ-calculus – Pseudo-Terms

Definitions:

- *Pre-pseudo-terms* Λ_Π^- are defined by the grammar

$$\Lambda_\Pi^- ::= V \mid (\Lambda_\Pi^- \, \Lambda_\Pi^-) \mid (\lambda x{:}\Pi.\Lambda_\Pi^-)$$

- Free variables $\mathrm{FV}(M)$, substitution $M[x := N]$ and α-equivalence $M \equiv_\alpha N$ are defined analogously to the untyped λ-calculus.

- Define the set of *pseudo terms* Λ_Π by

$$\Lambda_\Pi = \Lambda_\Pi / \equiv_\alpha = \{[M]_\alpha \mid M \in \Lambda_\Pi^-\}$$

Analogously to the untyped case, we identify pseudo-terms with pre-pseudo-terms.

Simply Typed λ-calculus – Typability

The (Church-style) *typability relation* $\Gamma \vdash M : \tau$ is defined inductively over M by the rules:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \text{ if } x \notin \mathrm{dom}(\Gamma) \qquad \frac{\Gamma, x : \sigma \vdash K : \tau}{\Gamma \vdash \lambda x{:}\sigma.K : \sigma \to \tau} \text{ if } x \notin \mathrm{dom}(\Gamma)$$

$$\frac{\Gamma \vdash K : \sigma \to \tau \qquad \Gamma \vdash L : \sigma}{\Gamma \vdash KL : \tau}$$

Definitions:

- M *has type* τ in Γ if $\Gamma \vdash M : \tau$.
- M is *typable* if there are τ and Γ such that $\Gamma \vdash M : \tau$.
- τ is *inhabited* if there are M and Γ such that $\Gamma \vdash M : \tau$.

Simply Typed λ-calculus – Sample terms

Some simply typed terms:

- combinator I: $\vdash \lambda x{:}\sigma.x : \sigma \rightarrow \sigma$
- combinator K: $\vdash \lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma \rightarrow \tau \rightarrow \sigma$
- combinator K*: $\vdash \lambda x{:}\sigma.\lambda y{:}\tau.y : \sigma \rightarrow \tau \rightarrow \tau$
- combinator S:
  $\vdash \lambda x{:}\sigma{\rightarrow}\tau{\rightarrow}\rho.\lambda y{:}\sigma{\rightarrow}\tau.\lambda z{:}\sigma.xz(yz) : (\sigma{\rightarrow}\tau{\rightarrow}\rho){\rightarrow}(\sigma{\rightarrow}\tau){\rightarrow}\sigma{\rightarrow}\rho$

Note that self-application (for instance $\lambda x.xx$) is **not** typable.

Simply Typed λ-calculus – β-reduction

Define β-reduction $\rightarrow_\beta$ analogous to the untyped case.

Properties of β-reduction:

- *Church-Rosser* property.
- *Subject reduction:* If $\Gamma \vdash M : \tau$ and $M \rightarrow_\beta N$ then $\Gamma \vdash N : \tau$.
- *Uniqueness of types:* If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$ then $\sigma = \tau$.
- *Weak normalization:* Every typable term M has a β-normalform.
- *Strong normalization:* No typable term M admits an infinite reduction sequence $M \rightarrow_\beta M' \rightarrow_\beta M'' \rightarrow_\beta \dots$
- Corollary: $\leftrightarrow_\beta^*$ is decidable for typable terms.
- Corollary: The simply typed λ-calculus is not Turing-complete.

Minimal Logic – Syntax

Minimal logic is the implicational fragment of intuitionistic propositional logic; formally:

- Assume an infinite alphabet of *propositional variables* PV, ranged over by A, B, C.
- *Formulas* Φ are defined by the grammar

$$\begin{array}{rcll} \Phi & ::= & PV & \textit{variable} \\ & \mid & (\Phi \to \Phi) & \textit{implication} \end{array}$$

  and ranged over by φ, ψ.

Notation: As usual (i. e., implication associates to the right).

Minimal Logic – Natural Deduction

Definitions:

- A *context* Δ is a finite subset of Φ.
- The *provability relation* $\Delta \vdash \varphi$ is defined inductively over φ by:
  - Assumption axiom $\dfrac{}{\Delta, \varphi \vdash \varphi}$
  - Arrow introduction $\dfrac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \to \psi}$
  - Arrow elimination $\dfrac{\Delta \vdash \varphi \to \psi \qquad \Delta \vdash \varphi}{\Delta \vdash \psi}$
- A *natural deduction proof* for φ (in context Δ) is a tree with root $\Delta \vdash \varphi$. Edges correspond to rule applications: leaf edges to the assumption axiom, other edges to introduction or elimination rules.

Minimal Logic – Sample Proofs

Two different proofs of the same formula:

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi, \psi \to \psi \vdash \psi \to \psi}}{\varphi, \psi \to \psi \vdash \varphi \to \psi \to \psi}\,(\mathrm{I})}{\varphi \vdash (\psi \to \psi) \to \varphi \to \psi \to \psi}\,(\mathrm{I}) \qquad \dfrac{\dfrac{}{\varphi, \psi \vdash \psi}}{\varphi \vdash \psi \to \psi}\,(\mathrm{I})}{\varphi \vdash \varphi \to \psi \to \psi}\,(\mathrm{E})
$$

$$
\dfrac{\dfrac{\dfrac{}{\varphi, \psi \vdash \psi}}{\varphi \vdash \psi \to \psi}\,(\mathrm{I})}{\varphi \vdash \varphi \to \psi \to \psi}\,(\mathrm{I})
$$

Minimal Logic – Proof Normalization

Long proof (prev. slide) in scope notation:

$$\begin{array}{|l} \varphi \\ \hline \begin{array}{|l} \psi \\ \hline \psi \end{array} \\ \psi \to \psi \\ \begin{array}{|l} \psi \to \psi \\ \hline \begin{array}{|l} \varphi \\ \hline \psi \to \psi \end{array} \\ \varphi \to \psi \to \psi \end{array} \\ (\psi \to \psi) \to \varphi \to \psi \to \psi \\ \varphi \to \psi \to \psi \end{array}$$

Proof normalized:

$$\begin{array}{|l} \varphi \\ \hline \begin{array}{|l} \psi \\ \hline \psi \end{array} \\ \psi \to \psi \\ \begin{array}{|l} \varphi \\ \hline \psi \to \psi \end{array} \\ \varphi \to \psi \to \psi \end{array}$$

Branches merged (short proof):

$$\begin{array}{|l} \varphi \\ \hline \begin{array}{|l} \varphi \\ \hline \begin{array}{|l} \psi \\ \hline \psi \end{array} \\ \psi \to \psi \end{array} \\ \varphi \to \psi \to \psi \end{array}$$

Curry-Howard Correspondence

Connection between simply typed λ calculus and natural deduction in minimal logic:

- Identify type variables U with propositional variable PV.
  Then types Π and formulas Φ coincide.
- From λ-calculus to minimal logic:
  If $\Gamma \vdash M : \tau$ then $|\Gamma| \vdash \tau$.
- From minimal logic to λ-calculus:
  If $\Delta \vdash \varphi$ then there are M and Γ such that $\Gamma \vdash M : \varphi$ and $\Gamma = \{x_\psi : \psi \mid \psi \in \Delta\}$.

Curry-Howard Correspondence – Summary

Simply typed λ-calculus	**Minimal logic**
type	formula
type variable	propositional variable
function type	implication
term	(constructive) proof
term variable	assumption
abstraction	arrow introduction
application	arrow elimination
inhabited type	provable formula
typable term	well-formed proof
β-reduction	proof normalization
β-redex	detour in proof
β-normal form	normal proof (no detours)

References

- Morten Heine Sørensen, Pawel Urzyczyn.
  Lectures on the Curry-Howard isomorphism.
  `http://www.diku.dk/forskning/topps/bibliography/1998.html`