Types in Programming Languages

Daniil Berezun

danya.berezun@gmail.com

2022

Outline for section 1

1. Simply Typed Lambda Calculus (STLC; $\lambda_\to$)
   - Syntax; à la Curry; à la Church
   - Properties
2. Curry–Howard Isomorphism
   - Intuitionistic Proposition Logic $Int_\to$
   - Curry–Howard Isomorphism
   - Hilbert's Propositional Calculus
   - KS-calculi
3. STLC
   - Strong Normalization
   - Type Inference
   - Robinson's Unification Algorithm
   - Set-Theoretic Semantics
4. Polymorphism
   - System F
   - Hindley–Milner Type System
   - Barendregt's Lambda Cube

Simple types

Base types Function type

$$\tau ::= \iota \mid \tau \to \tau$$

Simple types

Base types $\quad$ Function type

$$\tau ::= \iota \mid \tau \to \tau$$

Typing

à la Church (intrinsic, ontological)

- $\lambda x^{\alpha}.x \ : \ \alpha \to \alpha$
- syntax → typing → semantics
- $\lambda^{Ch}_{\to} ::= \lambda\, x^{\alpha}.\ \lambda^{Ch}_{\to} \mid \lambda^{Ch}_{\to}\ \lambda^{Ch}_{\to} \mid X$

$$\frac{}{x_i^{\alpha} \ : \ alpha}\ Var \qquad \frac{}{c_j^{\alpha} \ : \ alpha}\ Const$$

$$\frac{u \ : \ \beta}{\lambda x^{\alpha}.u \ : \ \alpha \to \beta}\ Abstr \qquad \frac{u \ : \ \alpha \to \beta \quad v \ : \ \alpha}{uv \ : \ \beta}\ App$$

à la Curry (extrinsic, semantical)

- $\vdash \lambda x.x \ : \ \alpha \to \alpha$
- syntax → semantics → typing
- $\lambda^{Cu}_{\to} ::= \Lambda = \lambda\, x.\Lambda \mid \Lambda\ \Lambda \mid X$

Simple types

Base types $\quad$ Function type

$$\tau ::= \iota \mid \tau \to \tau$$

Typing

à la Church (intrinsic, ontological)

- $\lambda x^{\alpha}.x \;:\; \alpha \to \alpha$
- syntax → typing → semantics
- $\lambda^{Ch}_{\to} ::= \lambda\, x^{\alpha}.\, \lambda^{Ch}_{\to} \mid \lambda^{Ch}_{\to}\; \lambda^{Ch}_{\to} \mid X$

$$\frac{}{x_i^{\alpha} \;:\; alpha}\; Var \qquad \frac{}{c_j^{\alpha} \;:\; alpha}\; Const$$

$$\frac{u \;:\; \beta}{\lambda x^{\alpha}.u \;:\; \alpha \to \beta}\; Abstr \qquad \frac{u \;:\; \alpha \to \beta \quad v \;:\; \alpha}{uv \;:\; \beta}\; App$$

à la Curry (extrinsic, semantical)

- $\vdash \lambda x.x \;:\; \alpha \to \alpha$
- syntax → semantics → typing
- $\lambda^{Cu}_{\to} ::= \Lambda = \lambda\, x.\Lambda \mid \Lambda\; \Lambda \mid X$

Term in context

$$\underbrace{x_1 : \alpha_1, \ldots, x_n : \alpha_n}_{\Gamma} \vdash u \;:\; \beta$$

Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \;:\; \alpha}\; Ax \qquad \frac{\Gamma \vdash u\; \alpha \to \beta \quad \Gamma \vdash v \;:\; \alpha}{\Gamma \vdash uv \;:\; \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u \;: \beta}{\Gamma \vdash \lambda x^{\alpha}.u \;:\; \alpha \to \beta} \to I$$

Examples

à la Curry	à la Church
$\lambda x.x \ : \ \alpha \to \alpha \qquad \lambda x.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \ : \ \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \ : \ \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \ : \ \alpha \to \beta \to \alpha$

Example

Examples

à la Curry	à la Church
$\lambda x.x \;:\; \alpha \to \alpha \qquad \lambda x.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \;:\; \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \;:\; \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \;:\; \alpha \to \beta \to \alpha$

Type Derivation

$$\dfrac{}{\vdash \lambda\, f^{(\beta \to \gamma)} g^{(\alpha \to \beta)} x^{\alpha}.\, f\,(g\; x) \;:\; (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

Examples

à la Curry	à la Church
$\lambda x.x \ : \ \alpha \to \alpha \qquad \lambda x.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \ : \ \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \ : \ \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \ : \ \alpha \to \beta \to \alpha$

Type Derivation

$$\dfrac{\dfrac{}{f : \beta \to \gamma \vdash \lambda\, g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \ : \ (\alpha \to \beta) \to \alpha \to \gamma} \to I}{\vdash \lambda\, f^{(\beta \to \gamma)} g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \ : \ (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

Examples

à la Curry	à la Church
$\lambda x.x \ : \ \alpha \to \alpha \qquad \lambda x.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \ : \ \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \ : \ \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \ : \ \alpha \to \beta \to \alpha$

Type Derivation

$$\dfrac{\dfrac{\overline{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\, x^{\alpha}.\ f\ (g\ x) \ : \ \alpha \to \gamma} \to I}{f : \beta \to \gamma \vdash \lambda\, g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \ : \ (\alpha \to \beta) \to \alpha \to \gamma} \to I}{\vdash \lambda\, f^{(\beta \to \gamma)} g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \ : \ (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

Example

Examples

à la Curry	à la Church
$\lambda x.x \ : \ \alpha \to \alpha \qquad \lambda x.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \ : \ \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \ : \ (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \ : \ \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \ : \ \alpha \to \beta \to \alpha$

Type Derivation

$$\dfrac{\dfrac{\dfrac{\overline{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\ (g\ x)\ :\ \gamma}\ \to E}{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\ x^{\alpha}.\ f\ (g\ x)\ :\ \alpha \to \gamma}\ \to I}{f : \beta \to \gamma \vdash \lambda\ g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x)\ :\ (\alpha \to \beta) \to \alpha \to \gamma}\ \to I}{\vdash \lambda\ f^{(\beta \to \gamma)} g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x)\ :\ (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}\ \to I$$

Examples

à la Curry	à la Church
$\lambda x.x \;:\; \alpha \to \alpha \qquad \lambda x.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \;:\; \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \;:\; \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \;:\; \alpha \to \beta \to \alpha$

Type Derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f \;:\; \beta \to \gamma}\,Ax \quad \dfrac{}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash g\ x \;:\; \beta}\,{\to}E}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\ (g\ x) \;:\; \gamma}\,{\to}E}{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\, x^{\alpha}.\ f\ (g\ x) \;:\; \alpha \to \gamma}\,{\to}I}{f : \beta \to \gamma \vdash \lambda\, g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \;:\; (\alpha \to \beta) \to \alpha \to \gamma}\,{\to}I}{\vdash \lambda\, f^{(\beta \to \gamma)} g^{(\alpha \to \beta)} x^{\alpha}.\ f\ (g\ x) \;:\; (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}\,{\to}I
$$

Example

Examples

à la Curry	à la Church
$\lambda x.x \;:\; \alpha \to \alpha \qquad \lambda x.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \;:\; \alpha \to \alpha \qquad \lambda x^{\alpha \to \beta}.x \;:\; (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \;:\; \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \;:\; \alpha \to \beta \to \alpha$

Type Derivation

$$
\dfrac{\dfrac{\dfrac{}{f:\beta\to\gamma,\ g:\alpha\to\beta,\ x:\alpha \vdash f \;:\; \beta\to\gamma}\,Ax \quad \dfrac{\dfrac{}{\cdots, g:\alpha\to\beta, \cdots \vdash g \;:\; \alpha\to\beta}\,Ax \quad \dfrac{}{\cdots,\ x:\alpha \vdash x \;:\; \beta}\,Ax}{f:\beta\to\gamma,\ g:\alpha\to\beta,\ x:\alpha \vdash g\ x \;:\; \beta}\,\to E}{\dfrac{f:\beta\to\gamma,\ g:\alpha\to\beta,\ x:\alpha \vdash f\ (g\ x) \;:\; \gamma}{\dfrac{f:\beta\to\gamma,\ g:\alpha\to\beta \vdash \lambda\, x^{\alpha}.\ f\ (g\ x) \;:\; \alpha\to\gamma}{\dfrac{f:\beta\to\gamma \vdash \lambda\, g^{(\alpha\to\beta)} x^{\alpha}.\ f\ (g\ x) \;:\; (\alpha\to\beta)\to\alpha\to\gamma}{\vdash \lambda\, f^{(\beta\to\gamma)} g^{(\alpha\to\beta)} x^{\alpha}.\ f\ (g\ x) \;:\; (\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma}\,\to I}\,\to I}\,\to I}\,\to E
$$

Correspondence between typing à la Curry and à la Church

Lemmas

Let $|\cdot| : \lambda_{\rightarrow}^{Ch} \rightarrow \Lambda \equiv \lambda_{\rightarrow}^{Cu}$, i.e. type annotation erasing map

1. $\forall U \in \lambda_{\rightarrow}^{Ch}. \quad \Gamma \vdash_{Ch} U : \tau \quad \Rightarrow \quad \Gamma \vdash_{Cu} |U| : \tau$
2. $\forall U \in \Lambda \equiv \lambda_{\rightarrow}^{Cu}. \quad \Gamma \vdash_{Cu} U : \tau \quad \Rightarrow \quad \exists V \in \lambda_{\rightarrow}^{Ch} : \quad \Gamma \vdash_{Ch} V : \tau \ \wedge |V| \equiv U$

Lemmas

Let $|\cdot|:\ \lambda_{\to}^{Ch} \to \Lambda \equiv \lambda_{\to}^{Cu}$, i.e. type annotation erasing map

1. $\forall U \in \lambda_{\to}^{Ch}.\quad \Gamma \vdash_{Ch} U\ :\ \tau \quad\Rightarrow\quad \Gamma \vdash_{Cu} |U|\ :\ \tau$
2. $\forall U \in \Lambda \equiv \lambda_{\to}^{Cu}.\quad \Gamma \vdash_{Cu} U : \tau \quad\Rightarrow\quad \exists V \in \lambda_{\to}^{Ch} :\quad \Gamma \vdash_{Ch} V : \tau\ \wedge |V| \equiv U$

Theorem

type is inhabited à la Curry $\Leftrightarrow$ type is inhabited à la Church

Correspondence between typing à la Curry and à la Church

Lemmas

Let $|\cdot|: \lambda_{\to}^{Ch} \to \Lambda \equiv \lambda_{\to}^{Cu}$, i.e. type annotation erasing map

1. $\forall U \in \lambda_{\to}^{Ch}. \quad \Gamma \vdash_{Ch} U : \tau \quad \Rightarrow \quad \Gamma \vdash_{Cu} |U| : \tau$
2. $\forall U \in \Lambda \equiv \lambda_{\to}^{Cu}. \quad \Gamma \vdash_{Cu} U : \tau \quad \Rightarrow \quad \exists V \in \lambda_{\to}^{Ch} : \quad \Gamma \vdash_{Ch} V : \tau \ \wedge |V| \equiv U$

Theorem

type is inhabited à la Curry $\Leftrightarrow$ type is inhabited à la Church

Standard Problems

Type Checking	$\vdash U : \tau ?$
Type Inference (assignment, synthesis)	$\vdash U : ?$
Type Inhabitation	$\vdash \ ? : \tau$

Lemmas

Let $|\cdot|: \lambda_{\rightarrow}^{Ch} \rightarrow \Lambda \equiv \lambda_{\rightarrow}^{Cu}$, i.e. type annotation erasing map

1. $\forall U \in \lambda_{\rightarrow}^{Ch}. \quad \Gamma \vdash_{Ch} U : \tau \quad \Rightarrow \quad \Gamma \vdash_{Cu} |U| : \tau$
2. $\forall U \in \Lambda \equiv \lambda_{\rightarrow}^{Cu}. \quad \Gamma \vdash_{Cu} U : \tau \quad \Rightarrow \quad \exists V \in \lambda_{\rightarrow}^{Ch} : \quad \Gamma \vdash_{Ch} V : \tau \ \wedge |V| \equiv U$

Theorem

type is inhabited à la Curry $\Leftrightarrow$ type is inhabited à la Church

Standard Problems

Type Checking	$\vdash U : \tau?$
Type Inference (assignment, synthesis)	$\vdash U : ?$
Type Inhabitation	$\vdash ? : \tau$

- All effectively solvable for STLC (both versions)
- In à-la Curry typechecking is equivalent to type inference:
  checking $UV : \tau?$ requires syntheses of $V :?$

Properties

Lemma [inversion, generation]

- $\Gamma \vdash x : \tau \Rightarrow x^{\tau} \in \Gamma$
- $\Gamma \vdash UV : \tau \Rightarrow \exists \sigma : \ \Gamma \vdash U : \sigma \to \tau \ \wedge \ \Gamma \vdash V : \sigma$
- $\Gamma \vdash \lambda x \ .U : \tau \Rightarrow \exists \sigma, \rho : \quad \Gamma, x : \sigma \vdash U : \rho \quad \wedge \quad \tau \equiv \sigma \to \rho$ [Curry]
- $\Gamma \vdash \lambda x^{\sigma}.U : \tau \Rightarrow \exists \quad \rho : \quad \Gamma, x : \sigma \vdash U : \rho \quad \wedge \quad \tau \equiv \sigma \to \rho$ [Church]

Lemma [Subterms are typable]

V — subterm U then $\quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \exists \Gamma', \tau' : \ \Gamma' \vdash V : \tau'$

Context Lemmas

[Thinning]

$$\left.\begin{array}{l} \Delta \subseteq \Gamma \\ \Delta \vdash U : \tau \end{array}\right\} \Rightarrow \Gamma \vdash U : \tau$$

[FV]

$$\Gamma \vdash U : \tau \Rightarrow FV(U) \subseteq \mathfrak{Dom}(\Gamma)$$

[Restriction]

$$\Gamma \vdash U : \tau \Rightarrow \Gamma|_{FV(U)} \vdash U : \tau$$

Lemma [inversion, generation]

- $\Gamma \vdash x : \tau \Rightarrow x^{\tau} \in \Gamma$
- $\Gamma \vdash UV : \tau \Rightarrow \exists \sigma : \ \Gamma \vdash U : \sigma \to \tau \ \wedge \ \Gamma \vdash V : \sigma$
- $\Gamma \vdash \lambda x \ .U : \tau \Rightarrow \exists \sigma, \rho : \quad \Gamma, x : \sigma \vdash U : \rho \quad \wedge \quad \tau \equiv \sigma \to \rho$ [Curry]
- $\Gamma \vdash \lambda x^{\sigma}.U : \tau \Rightarrow \exists \quad \rho : \quad \Gamma, x : \sigma \vdash U : \rho \quad \wedge \quad \tau \equiv \sigma \to \rho$ [Church]

Lemma [Subterms are typable]

V — subterm U then $\quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \exists \Gamma', \tau' : \ \Gamma' \vdash V : \tau'$

Context Lemmas

[Thinning]

$$\left.\begin{array}{l} \Delta \subseteq \Gamma \\ \Delta \vdash U : \tau \end{array}\right\} \Rightarrow \Gamma \vdash U : \tau$$

[FV]

$$\Gamma \vdash U : \tau \Rightarrow FV(U) \subseteq \mathfrak{Dom}(\Gamma)$$

[Restriction]

$$\Gamma \vdash U : \tau \Rightarrow \Gamma|_{FV(U)} \vdash U : \tau$$

Example: we can prove that some (pre-)terms has no type

$$\Gamma \nvdash xx : \tau \quad \nvdash \omega : \tau \quad \nvdash \Omega : \tau \quad \nvdash \mathbb{Y} : \tau$$

Lemma [substitution]

$$\frac{\Gamma, x : \sigma \vdash U : \tau \quad \Gamma \vdash V : \sigma}{\Gamma \vdash U[x/V] : \tau}$$

Theorem [type preserving]

$$U \twoheadrightarrow_\beta V \quad \wedge \quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \Gamma \vdash V : \tau$$

Lemma [substitution]

$$\frac{\Gamma, x : \sigma \vdash U : \tau \quad \Gamma \vdash V : \sigma}{\Gamma \vdash U[x/V] : \tau}$$

Theorem [type preserving]

$$U \twoheadrightarrow_\beta V \quad \wedge \quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \Gamma \vdash V : \tau$$

Conclusion

$\lambda_\rightarrow$ is closed under β-reduction

NB: reverse is not true [β-reduction may lose some information]

- β-reduction can turn untypable term into typable one:

Lemma [substitution]

$$\frac{\Gamma, x : \sigma \vdash U : \tau \quad \Gamma \vdash V : \sigma}{\Gamma \vdash U[x/V] : \tau}$$

Theorem [type preserving]

$$U \twoheadrightarrow_\beta V \quad \wedge \quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \Gamma \vdash V : \tau$$

Conclusion

$\lambda_\to$ is closed under β-reduction

NB: reverse is not true [β-reduction may lose some information]

- β-reduction can turn untypable term into typable one:

$$\nvdash \omega \qquad (\lambda x.y)\omega \to_\beta y \qquad \Gamma = y : \tau_1 \vdash y : \tau_1 \qquad \nvdash (\lambda x.y)\omega : \tau_1$$

- β-reduction can make type more general:

Lemma [substitution]

$$\frac{\Gamma, x : \sigma \vdash U : \tau \quad \Gamma \vdash V : \sigma}{\Gamma \vdash U[x/V] : \tau}$$

Theorem [type preserving]

$$U \twoheadrightarrow_\beta V \quad \wedge \quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \Gamma \vdash V : \tau$$

Conclusion

$\lambda_\to$ is closed under β-reduction

NB: reverse is not true [β-reduction may lose some information]

- β-reduction can turn untypable term into typable one:

$$\nvdash \omega \qquad (\lambda x.y)\omega \to_\beta y \qquad \Gamma = y : \tau_1 \vdash y : \tau_1 \qquad \nvdash (\lambda x.y)\omega : \tau_1$$

- β-reduction can make type more general:

$$\lambda x.\lambda z.((\lambda y.z)(zx)) : \alpha \to (\alpha \to \beta) \to (\alpha \to \beta) \quad \to_\beta \quad \lambda x.\lambda z.z : \alpha \to (\tau \to \tau)$$

Lemma [substitution]

$$\frac{\Gamma, x : \sigma \vdash U : \tau \quad \Gamma \vdash V : \sigma}{\Gamma \vdash U[x/V] : \tau}$$

Theorem [type preserving]

$$U \twoheadrightarrow_\beta V \quad \wedge \quad \Gamma \vdash U : \tau \quad \Rightarrow \quad \Gamma \vdash V : \tau$$

Conclusion

$\lambda_\to$ is closed under β-reduction

NB: reverse is not true [β-reduction may lose some information]

- β-reduction can turn untypable term into typable one:

$$\nvdash \omega \qquad (\lambda x.y)\omega \to_\beta y \qquad \Gamma = y : \tau_1 \vdash y : \tau_1 \qquad \nvdash (\lambda x.y)\omega : \tau_1$$

- β-reduction can make type more general:

$$\lambda x.\lambda z.((\lambda y.z)(zx)) : \alpha \to (\alpha \to \beta) \to (\alpha \to \beta) \quad \to_\beta \quad \lambda x.\lambda z.z : \alpha \to (\tau \to \tau)$$

Lemma [Type substitution]

$\Gamma \vdash U : \sigma \Rightarrow \Gamma[\alpha/\tau] \vdash U : \sigma[\alpha/\tau]$ [*Curry*] $\qquad \Gamma \vdash U : \sigma \Rightarrow \Gamma[\alpha/\tau] \vdash U[\alpha/\tau] : \sigma[\alpha/\tau]$ [*Church*]

Outline for section 2

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\ Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v \ : \ \alpha}{\Gamma \vdash uv \ : \ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u \ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u \ : \ \alpha \to \beta} \to I$$

Let’s erase terms!

Curry–Howard Isomorphism (Correspondence)

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\ Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v \ : \ \alpha}{\Gamma \vdash uv \ : \ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u \ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u \ : \ \alpha \to \beta} \to I$$

Let's erase terms!

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$

Curry–Howard Isomorphism (Correspondence)

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\ Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v\ :\ \alpha}{\Gamma \vdash uv\ :\ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u\ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u\ :\ \alpha \to \beta} \to I$$

Let's erase terms! **Implicative part of Intuitionistic logic** $(Int_{\to})$

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$

Modus ponens

Hilbert's Deduction Theorem

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\ Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v \ : \ \alpha}{\Gamma \vdash uv \ : \ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u \ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u \ : \ \alpha \to \beta} \to I$$

Let's erase terms! **Implicative part of Intuitionistic logic** $(Int_{\to})$

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$

Modus ponens

Hilbert's Deduction Theorem

- No *excluded middle*: $\vDash \alpha \vee \neg\alpha$
- No *Price law*: $\vDash ((\alpha \to \beta) \to \alpha) \to \alpha$
- No *double negation*: $\vDash \alpha \leftrightarrow \neg\neg\alpha$

Curry–Howard Isomorphism (Correspondence)

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \;:\; \alpha}\;Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v \;:\; \alpha}{\Gamma \vdash uv \;:\; \beta}\;\to E \qquad \frac{\Gamma, x : \alpha \vdash u \;: \beta}{\Gamma \vdash \lambda x^{\alpha}.u \;:\; \alpha \to \beta}\;\to I$$

Let's erase terms! **Implicative part of Intuitionistic logic** $(Int_{\to})$

$$\frac{}{\Gamma, \alpha \vdash \alpha}\;Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta}\;\to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta}\;\to I$$

Modus ponens

Hilbert's Deduction Theorem

- No *excluded middle*: $\vDash \alpha \vee \neg\alpha$
- No *Price law*: $\vDash ((\alpha \to \beta) \to \alpha) \to \alpha$
- No *double negation*: $\vDash \alpha \leftrightarrow \neg\neg\alpha$
- Is $\sqrt{2}^{\sqrt{2}}$ irrational?

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f \;:\; \beta \to \gamma}\;Ax \qquad \dfrac{\dfrac{}{\Gamma \vdash g \;:\; \alpha \to \beta}\;Ax \qquad \dfrac{}{\Gamma \vdash x \;:\; \beta}\;Ax}{\Gamma \vdash g\ x \;:\; \beta}\;\to E}{\Gamma \equiv f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\,(g\ x) \;:\; \gamma}\;\to E}{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\, x.\ f\,(g\ x) \;:\; \alpha \to \gamma}\;\to I}{f : \beta \to \gamma \vdash \lambda\, gx.\ f\,(g\ x) \;:\; (\alpha \to \beta) \to \alpha \to \gamma}\;\to I}{\vdash \lambda\, fgx.\ f\,(g\ x) \;:\; (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}\;\to I$$

Curry–Howard Isomorphism (Correspondence)

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\,Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v \ : \ \alpha}{\Gamma \vdash uv \ : \ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u \ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u \ : \ \alpha \to \beta} \to I$$

Let's erase terms! **Implicative part of Intuitionistic logic** ($Int_{\to}$)

$$\frac{}{\Gamma, \alpha \vdash \alpha}\,Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$

Modus ponens

Hilbert's Deduction Theorem

- No *excluded middle*: $\vDash \alpha \vee \neg\alpha$
- No *Price law*: $\vDash ((\alpha \to \beta) \to \alpha) \to \alpha$
- No *double negation*: $\vDash \alpha \leftrightarrow \neg\neg\alpha$
- Is $\sqrt{2}^{\sqrt{2}}$ irrational?

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f \ : \ \beta \to \gamma}\,Ax \quad \dfrac{\dfrac{}{\Gamma \vdash g \ : \ \alpha \to \beta}\,Ax \quad \dfrac{}{\Gamma \vdash x \ : \ \beta}\,Ax}{\Gamma \vdash g\ x \ : \ \beta} \to E}{\Gamma \equiv f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\,(g\ x) \ : \ \gamma} \to E}{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\, x.\ f\,(g\ x) \ : \ \alpha \to \gamma} \to I}{f : \beta \to \gamma \vdash \lambda\, gx.\ f\,(g\ x) \ : \ (\alpha \to \beta) \to \alpha \to \gamma} \to I}{\vdash \lambda\, fgx.\ f\,(g\ x) \ : \ (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

Remind:Typing rules

$$\frac{}{\Gamma, x : \alpha \vdash x \ : \ \alpha}\ Ax \qquad \frac{\Gamma \vdash u\ \alpha \to \beta \quad \Gamma \vdash v\ :\ \alpha}{\Gamma \vdash uv\ :\ \beta} \to E \qquad \frac{\Gamma, x : \alpha \vdash u\ : \beta}{\Gamma \vdash \lambda x^{\alpha}.u\ :\ \alpha \to \beta} \to I$$

Let's erase terms! **Implicative part of Intuitionistic logic** ($Int_{\to}$)

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ Ax \qquad \frac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$

Modus ponens

Hilbert's Deduction Theorem

- No *excluded middle*: $\vDash \alpha \vee \neg\alpha$
- No *Price law*: $\vDash ((\alpha \to \beta) \to \alpha) \to \alpha$
- No *double negation*: $\vDash \alpha \leftrightarrow \neg\neg\alpha$
- Is $\sqrt{2}^{\sqrt{2}}$ irrational?

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \beta \to \gamma}\ Ax \quad \dfrac{\dfrac{}{\Gamma \vdash \alpha \to \beta}\ Ax \quad \dfrac{}{\Gamma \vdash \beta}\ Ax}{\Gamma \vdash \beta} \to E}{\Gamma \equiv \beta \to \gamma,\ \alpha \to \beta,\ \alpha \vdash \gamma} \to E}{\beta \to \gamma,\ \alpha \to \beta \vdash \alpha \to \gamma} \to I}{\beta \to \gamma \vdash (\alpha \to \beta) \to \alpha \to \gamma} \to I}{\vdash (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

Curry–Howard Isomorphism (Correspondence)

Programming	Logic
Type	Proposition
Term	Prove
Type inhabitance	Provability
Function type	Implication
Pair	Conjunction
Sum type	Disjunction
Unit type	True
Void	False
Π-type	$\forall$
$\sum$-type	$\exists$
$\dots$	$\dots$

Curry–Howard Isomorphism (Correspondence)

Programming	Logic
Type	Proposition
Term	Prove
Type inhabitance	Provability
Function type	Implication
Pair	Conjunction
Sum type	Disjunction
Unit type	True
Void	False
Π-type	$\forall$
$\sum$-type	$\exists$
...	...

Lemma 1 [$Int_{\rightarrow} \Rightarrow \lambda_{\rightarrow}$]

If $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash_{\lambda_{\rightarrow}}^{Curry} u : \beta$
then $\alpha_1, \ldots, \alpha_n \vdash_{Int_{\rightarrow}} \beta$

Lemma 2 [$\lambda_{\rightarrow} \Rightarrow Int_{\rightarrow}$]

If $\alpha_1, \ldots, \alpha_n \vdash_{Int_{\rightarrow}} \beta$
then $\exists u, x_1, \ldots, x_n : \quad x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash_{\lambda_{\rightarrow}}^{Curry} u : \beta$

Programming	Logic
Type	Proposition
Term	Prove
Type inhabitance	Provability
Function type	Implication
Pair	Conjunction
Sum type	Disjunction
Unit type	True
Void	False
Π-type	$\forall$
$\sum$-type	$\exists$
...	...

Lemma 1 [$Int_{\rightarrow} \Rightarrow \lambda_{\rightarrow}$]

If $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash_{\lambda_{\rightarrow}}^{Curry} u : \beta$
then $\alpha_1, \ldots, \alpha_n \vdash_{Int_{\rightarrow}} \beta$

Lemma 2 [$\lambda_{\rightarrow} \Rightarrow Int_{\rightarrow}$]

If $\alpha_1, \ldots, \alpha_n \vdash_{Int_{\rightarrow}} \beta$
then $\exists u, x_1, \ldots, x_n : \quad x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash_{\lambda_{\rightarrow}}^{Curry} u : \beta$

Note: u is not unique here:

Which α?

$$\dfrac{\dfrac{\dfrac{}{\alpha, \alpha \vdash \alpha}}{\alpha \vdash \alpha \rightarrow \alpha}}{\vdash \alpha \rightarrow (\alpha \rightarrow \alpha)}$$

$\lambda x.\lambda y.x : \ \alpha \rightarrow \alpha \rightarrow \alpha$
$\lambda x.\lambda y.y : \ \alpha \rightarrow \alpha \rightarrow \alpha$

Hilbert's propositional calculus

Axioms

The Inference Rule

$$A \to (B \to A)[A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C))[A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Hilbert's propositional calculus

Axioms

The Inference Rule

$A \to (B \to A)[A_1]$ $\quad (A \to (B \to C)) \to ((A \to B) \to (A \to C))[A_2]$ $\quad \dfrac{A \quad A \to B}{B}\ [MP]$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

Hilbert's propositional calculus

Axioms

The Inference Rule

$$A \to (B \to A)[A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C))[A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

Alternative syntax (derivation):

Hilbert's propositional calculus

Axioms The Inference Rule

$$A \to (B \to A) [A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C)) [A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

1. $A \to (\overbrace{(A \to A)}^{B} \to A)$ $A1$

Alternative syntax (derivation):

$$\frac{}{A \to ((A \to A) \to A)}\,A1$$

Hilbert's propositional calculus

Axioms — The Inference Rule

$$A \to (B \to A)[A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C))[A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

1. $A \to (\overbrace{(A \to A)}^{B} \to A)$ $\qquad A1$
2. $(A \to (\overbrace{(A \to A)}^{B} \to \overbrace{A}^{C})) \to ((A \to \overbrace{(A \to A)}^{B}) \to (A \to \overbrace{A}^{C}))$ $\qquad A2$

Alternative syntax (derivation):

$$\frac{}{A \to ((A \to A) \to A)}\,A1 \qquad \frac{}{(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))}\,A2$$

Hilbert's propositional calculus

Axioms

The Inference Rule

$$A \to (B \to A) [A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C)) [A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

1. $A \to (\overbrace{(A \to A)}^{B} \to A)$ $\quad A1$
2. $(A \to (\overbrace{(A \to A)}^{B} \to \overbrace{A}^{C})) \to ((A \to \overbrace{(A \to A)}^{B}) \to (A \to \overbrace{A}^{C}))$ $\quad A2$
3. $A \to (A \to A) \to (A \to A)$ $\quad MP$ 1. 2.

Alternative syntax (derivation):

$$\dfrac{\dfrac{}{A \to ((A \to A) \to A)}\,A1 \qquad \dfrac{}{(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))}\,A2}{A \to (A \to A) \to (A \to A)}\,MP$$

Hilbert's propositional calculus

Axioms

The Inference Rule

$$A \to (B \to A) [A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C)) [A_2] \qquad \frac{A \quad A \to B}{B} [MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

1. $A \to (\overbrace{(A \to A)}^{B} \to A)$ — $A1$
2. $(A \to (\overbrace{(A \to A)}^{B} \to \overbrace{A}^{C})) \to ((A \to \overbrace{(A \to A)}^{B}) \to (A \to \overbrace{A}^{C}))$ — $A2$
3. $A \to (A \to A) \to (A \to A)$ — MP 1. 2.
4. $A \to (A \to A)$ — $A1$

Alternative syntax (derivation):

$$\frac{}{A \to (A \to A)} A1 \qquad \frac{\dfrac{}{A \to ((A \to A) \to A)} A1 \quad \dfrac{}{(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))} A2}{A \to (A \to A) \to (A \to A)} MP$$

Hilbert's propositional calculus

Axioms — The Inference Rule

$$A \to (B \to A) [A_1] \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C)) [A_2] \qquad \frac{A \quad A \to B}{B}\,[MP]$$

- Classical logic: just add the Pierce law: $((A \to B) \to A) \to A$

Example: $\vdash_{pc} A \to A$

1. $A \to (\overbrace{(A \to A)}^{B} \to A)$ — $A1$
2. $(A \to (\overbrace{(A \to A)}^{B} \to \overbrace{A}^{C})) \to ((A \to \overbrace{(A \to A)}^{B}) \to (A \to \overbrace{A}^{C}))$ — $A2$
3. $A \to (A \to A) \to (A \to A)$ — MP 1. 2.
4. $A \to (A \to A)$ — $A1$
5. $A \to A$ — MP 3. 4

Alternative syntax (derivation):

$$\dfrac{\dfrac{}{A \to (A \to A)}\,A1 \qquad \dfrac{\dfrac{}{A \to ((A \to A) \to A)}\,A1 \qquad \dfrac{}{(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))}\,A2}{A \to (A \to A) \to (A \to A)}\,MP}{A \to A}\,MP$$

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Hence, PC $\sim Int_{\to}$

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Hence, PC $\sim Int_{\to}$

KS-calculus

- IPC $\sim CL_{\to}$ (aka KS-calculus)

$A \to (B \to A)$	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	$\dfrac{A \quad A \to B}{A}$
$K: \ \alpha \to (\beta \to \alpha)$	$S: \ (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	application

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Hence, PC $\sim Int_{\to}$

KS-calculus

- IPC $\sim CL_{\to}$ (aka $\mathbb{KS}$-calculus)

$A \to (B \to A)$	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	$\dfrac{A \quad A \to B}{A}$
$K:\ \alpha \to (\beta \to \alpha)$	$S:\ (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	application

- Terms: $T^{\tau} ::= v_i \mid K_{\alpha,\beta} \mid S_{\alpha,\beta,\gamma} \mid T^{\alpha\to\beta}\ T^{\alpha}$
- Reductions: $K_{\alpha,\beta} x^{\alpha} y^{\beta} \to x^{\alpha} \qquad S_{\alpha,\beta,\gamma} x^{\alpha\to(\beta\to\gamma)} y^{\alpha\to\beta} z^{\gamma} \to xz(yz)$ where $x, y, z \in T$

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Hence, PC $\sim Int_{\to}$

KS-calculus

- IPC $\sim CL_{\to}$ (aka KS-calculus)

$A \to (B \to A)$	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	$\dfrac{A \quad A \to B}{A}$
$K: \ \alpha \to (\beta \to \alpha)$	$S: \ (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	application

- Terms: $T^{\tau} ::= v_i \mid K_{\alpha,\beta} \mid S_{\alpha,\beta,\gamma} \mid T^{\alpha\to\beta}\ T^{\alpha}$
- Reductions: $K_{\alpha,\beta} x^{\alpha} y^{\beta} \to x^{\alpha}$ $\quad S_{\alpha,\beta,\gamma} x^{\alpha\to(\beta\to\gamma)} y^{\alpha\to\beta} z^{\gamma} \to xz(yz)$ $\quad$ where $x, y, z \in T$
- Interpretation of $CL_{\to}$ in $\lambda_{\to}$: $K^{\lambda_{\to}}_{\alpha,\beta} \leftrightharpoons \lambda x^{\alpha} y^{\beta}.x$ $\quad S^{\lambda_{\to}}_{\alpha,\beta,\gamma} \leftrightharpoons \lambda x\ y\ z.\ x\ z\ (y\ z)$

Theorem [Hilbert's Deduction Theorem]

$$\Gamma, A \vdash_{pc} B \quad \Leftrightarrow \quad \Gamma \vdash_{pc} A \to B$$

Hence, PC $\sim Int_{\to}$

KS-calculus

- IPC $\sim CL_{\to}$ (aka KS-calculus)

$A \to (B \to A)$	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	$\dfrac{A \quad A \to B}{A}$
$K: \ \alpha \to (\beta \to \alpha)$	$S: \ (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	application

- Terms: $T^{\tau} ::= v_i \mid K_{\alpha,\beta} \mid S_{\alpha,\beta,\gamma} \mid T^{\alpha\to\beta}\ T^{\alpha}$
- Reductions: $K_{\alpha,\beta} x^{\alpha} y^{\beta} \to x^{\alpha}$ $\quad S_{\alpha,\beta,\gamma} x^{\alpha\to(\beta\to\gamma)} y^{\alpha\to\beta} z^{\gamma} \to xz(yz)$ $\quad$ where $x, y, z \in T$
- Interpretation of $CL_{\to}$ in $\lambda_{\to}$: $K^{\lambda_{\to}}_{\alpha,\beta} \leftrightharpoons \lambda x^{\alpha} y^{\beta}.x$ $\quad S^{\lambda_{\to}}_{\alpha,\beta,\gamma} \leftrightharpoons \lambda x\ y\ z.\ x\ z\ (y\ z)$

Theorem

$$M \to_{CL_{\to}} N \quad \Rightarrow \quad M^{\lambda_{\to}} \twoheadrightarrow_{\beta} N^{\lambda_{\to}}$$

Theorem [Abstraction in $CL_{\to}$]

$\forall M \in CL_{\to}.\ \forall$ var $x : \alpha.\ \exists$ term $\lambda^* x^\alpha.M$ s.t. $x \notin FV(\lambda^* x^\alpha.M)$ and $(\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_{\to}} M[x^\alpha/N^\alpha]$

Theorem [Abstraction in $CL_{\to}$]

$$\forall M \in CL_{\to}.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_{\to}} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\to\alpha,\alpha} K_{\alpha,\alpha\to\alpha} K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$)

Theorem [Abstraction in $CL_\to$]

$$\forall M \in CL_\to.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_\to} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\to\alpha,\alpha}K_{\alpha,\alpha\to\alpha}K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\qquad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha}y^\beta$

Theorem [Abstraction in $CL_\to$]

$$\forall M \in CL_\to.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_\to} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\to\alpha,\alpha}K_{\alpha,\alpha\to\alpha}K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\qquad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha}y^\beta$

$$\lambda^* x^\alpha.(M^{\beta\to\gamma}N^\gamma) \leftrightharpoons S_{\alpha,\beta,\gamma}(\lambda^* x^\alpha.M)(\lambda^* x^\alpha.N)$$

Theorem [Abstraction in $CL_{\to}$]

$$\forall M \in CL_{\to}.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_{\to}} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\to\alpha,\alpha} K_{\alpha,\alpha\to\alpha} K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\qquad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha} y^\beta$

$$\lambda^* x^\alpha.(M^{\beta\to\gamma} N^\gamma) \leftrightharpoons S_{\alpha,\beta,\gamma}(\lambda^* x^\alpha.M)(\lambda^* x^\alpha.N)$$

Theorem [completeness]

$$\forall M \in CL_{\to},\ \{x_1, \ldots, x_n\} = FV(M)\ \exists \text{ closed term (combinator) } N :\ N\ x_1 \ldots x_n \twoheadrightarrow_{CL_{\to}} M$$

Proof: $\qquad N = \lambda^* x_1.\ \ldots \lambda^* x_n.M$

$CL_{\rightarrow}$ Properties

Theorem [Abstraction in $CL_{\rightarrow}$]

$$\forall M \in CL_{\rightarrow}.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_{\rightarrow}} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\rightarrow\alpha,\alpha}K_{\alpha,\alpha\rightarrow\alpha}K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\quad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha}y^\beta$

$$\lambda^* x^\alpha.(M^{\beta\rightarrow\gamma}N^\gamma) \leftrightharpoons S_{\alpha,\beta,\gamma}(\lambda^* x^\alpha.M)(\lambda^* x^\alpha.N)$$

Theorem [completeness]

$$\forall M \in CL_{\rightarrow},\ \{x_1, \ldots, x_n\} = FV(M)\ \exists \text{ closed term (combinator) } N :\ N\ x_1 \ldots x_n \twoheadrightarrow_{CL_{\rightarrow}} M$$

Proof: $\quad N = \lambda^* x_1.\ \ldots \lambda^* x_n.M$

Is there any difference between λ^* and λ (abstraction in STLC)?

Theorem [Abstraction in $CL_\rightarrow$]

$$\forall M \in CL_\rightarrow.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_\rightarrow} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\to\alpha,\alpha} K_{\alpha,\alpha\to\alpha} K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\quad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha} y^\beta$

$$\lambda^* x^\alpha.(M^{\beta\to\gamma} N^\gamma) \leftrightharpoons S_{\alpha,\beta,\gamma}(\lambda^* x^\alpha.M)(\lambda^* x^\alpha.N)$$

Theorem [completeness]

$$\forall M \in CL_\rightarrow,\ \{x_1, \ldots, x_n\} = FV(M)\ \exists \text{ closed term (combinator) } N :\ N\ x_1 \ldots x_n \twoheadrightarrow_{CL_\rightarrow} M$$

Proof: $\quad N = \lambda^* x_1.\ \ldots \lambda^* x_n.M$

Is there any difference between λ^* and λ (abstraction in STLC)? YES!

$$M = M' \text{ in } CL_\rightarrow \quad \not\Rightarrow \quad \lambda^* x.M = \lambda^* x.N \text{ in } CL_\rightarrow$$

Counterexample: $K\ x\ K =_{CL} x$ but there are two different normal forms:

$$\lambda^* x.K\ x\ K = S(S(KK)(S(KK)))(KK) \qquad \lambda^* x.x = SKK$$

$CL_{\rightarrow}$ Properties

Theorem [Abstraction in $CL_{\rightarrow}$]

$$\forall M \in CL_{\rightarrow}.\ \forall \text{ var } x : \alpha.\ \exists \text{ term } \lambda^* x^\alpha.M \text{ s.t. } x \notin FV(\lambda^* x^\alpha.M) \text{ and } (\lambda^* x^\alpha.M)N^\alpha \twoheadrightarrow_{CL_{\rightarrow}} M[x^\alpha/N^\alpha]$$

Proof: $\lambda^* x^\alpha.x^\alpha \leftrightharpoons S_{\alpha,\alpha\rightarrow\alpha,\alpha}K_{\alpha,\alpha\rightarrow\alpha}K_{\alpha,\alpha}$ (i.e. $I \equiv SKK$) $\qquad \lambda^* x^\alpha.y^\beta \leftrightharpoons K_{\beta,\alpha}y^\beta$

$$\lambda^* x^\alpha.(M^{\beta\rightarrow\gamma}N^\gamma) \leftrightharpoons S_{\alpha,\beta,\gamma}(\lambda^* x^\alpha.M)(\lambda^* x^\alpha.N)$$

Theorem [completeness]

$$\forall M \in CL_{\rightarrow},\ \{x_1, \ldots, x_n\} = FV(M)\ \exists \text{ closed term (combinator) } N :\ N\ x_1 \ldots x_n \twoheadrightarrow_{CL_{\rightarrow}} M$$

Proof: $\qquad N = \lambda^* x_1.\ \ldots \lambda^* x_n.M$

Is there any difference between λ^* and λ (abstraction in STLC)? YES!

$$M = M' \text{ in } CL_{\rightarrow} \quad \not\Rightarrow \quad \lambda^* x.M = \lambda^* x.N \text{ in } CL_{\rightarrow}$$

Counterexample: $K\ x\ K =_{CL} x$ but there are two different normal forms:

$$\lambda^* x.K\ x\ K = S(S(KK)(S(KK)))(KK) \qquad \lambda^* x.x = SKK$$

Untyped CL

Two terms Equality Derivability Problem is undecidable

CL is a “simplest” system with so “simple” undesidable problem

Definition [computable terms] [$\leftrightharpoons$ *Comp*]

- $\alpha = p \quad u \in Comp_P \Leftrightarrow u \in SN$
- $\alpha = \sigma \to \tau \quad u \in Comp_{\sigma\to\tau} \Leftrightarrow \forall v \in Comp_\sigma : \; uv \in Comp_\tau$

Lemma 1

(a) $u \in Comp_\tau \quad \Rightarrow \quad u \in SN$

(b) $u \in Comp_\tau \wedge u \to_\beta u' \Rightarrow u' \in Comp_\tau$

(c) $\left.\begin{array}{l} u : \tau, u \not\equiv \lambda \ldots \\ \forall u'.(u \to u' \Rightarrow u' \in Comp_\tau) \end{array}\right\} \Rightarrow u \in Comp_\tau$

Lemma 2

$$\forall v \in Comp_\sigma.\ u[x/v] \in Comp_\tau \Rightarrow \lambda x.u \in Comp_{\sigma\to\tau}$$

Lemma 3

$$\left.\begin{array}{l} FV(u) \subseteq \{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\} \\ \forall i.v_i \in Comp_{\sigma_i} \end{array}\right\} \Rightarrow u[x_1/v_1, \ldots, x_n/v_n] \in Comp_\tau$$

Theorem [Strong Normalization]

$u : \tau \Rightarrow u \in Comp_\tau \qquad [\overset{def}{\Rightarrow} u \in SN]$

Theorem [STLC: WN + CR]

$\forall u \; \exists! u_0 \in NF : \; u =_\beta u_0$

Definition [principal (most general type)]

τ is *most general type* of a term in context Γ if both

- $\Gamma \vdash u : \tau$
- $\Gamma \vdash u : \tau' \quad \Rightarrow \quad \tau' = \sigma(\tau)$ where
  σ is a type substitution that does not affect types from context Γ

Type Inference: Principal Type

Definition [principal (most general type)]

τ is *most general type* of a term in context Γ if both

- $\Gamma \vdash u : \tau$
- $\Gamma \vdash u : \tau' \quad \Rightarrow \quad \tau' = \sigma(\tau)$ where
  σ is a type substitution that does not affect types from context Γ

Example

- $\vdash \lambda x.\lambda y.x : \ \alpha_1 \to \alpha_2 \to \alpha_1 \qquad \alpha_1 := \beta,\ \alpha_2 := \gamma \qquad \beta \to \gamma \to \beta$ — *ok*
- $x : \alpha \vdash x : \alpha \qquad \alpha := \beta \to \gamma \qquad x : \alpha \vdash x : \ \beta \to \gamma$ — *not ok*

Type Inference: Principal Type

Definition [principal (most general type)]

τ is *most general type* of a term in context Γ if both

- $\Gamma \vdash u : \tau$
- $\Gamma \vdash u : \tau' \quad \Rightarrow \quad \tau' = \sigma(\tau)$ where
  σ is a type substitution that does not affect types from context Γ

Example

- $\vdash \lambda x.\lambda y.x : \ \alpha_1 \to \alpha_2 \to \alpha_1 \qquad \alpha_1 := \beta,\ \alpha_2 := \gamma \qquad \beta \to \gamma \to \beta$ — *ok*
- $x : \alpha \vdash x : \alpha \qquad \alpha := \beta \to \gamma \qquad x : \alpha \vdash x : \ \beta \to \gamma$ — *not ok*

Lemma

$\Gamma \vdash u : \beta$ and substitution σ doesn't affect primitive types (constants) from Γ
then $\Gamma \vdash u : \sigma(\beta)$

Theorem [Type inference is possible]

Term u is typable in context Γ then exist the principal type u in context Γ and an efficient algorithm of its inference

Rule	Equations
$\Gamma, x : \alpha \vdash x : r$	$r \approx \alpha$
$\dfrac{\Gamma \vdash u : r_2 \quad \Gamma \vdash v : r_3}{\Gamma \vdash uv : r_1}$	$r_2 \approx r_3 \rightarrow r_1$
$\dfrac{\Gamma, x : r_2 \vdash u : r_3}{\Gamma \vdash \lambda x.u : r_1}$	$r_1 \approx r_2 \rightarrow r_3$

Type Inference: Unification

Rule	Equations
$\Gamma, x : \alpha \vdash x : r$	$r \approx \alpha$
$\dfrac{\Gamma \vdash u : r_2 \quad \Gamma \vdash v : r_3}{\Gamma \vdash uv : r_1}$	$r_2 \approx r_3 \to r_1$
$\dfrac{\Gamma, x : r_2 \vdash u : r_3}{\Gamma \vdash \lambda x.u : r_1}$	$r_1 \approx r_2 \to r_3$

Definition [Unifier]

A *unifier* for equation system $\begin{cases} \alpha_1 \approx \beta_1 \\ \dots \\ \alpha_n \approx \beta_n \end{cases}$
is a substitution $\sigma :\quad \forall i.\ \sigma(\alpha_i) = \sigma(\beta_i)$

Example

$r_1 \to (r_2 \to r_3) \approx (r_4 \to r_5) \to r_6$
$r_1 := \alpha \to \alpha$
$r_6 := \alpha \to \alpha$
$r_{2,3,4,5} := \alpha$
$(\alpha \to \alpha) \to (\alpha \to \alpha)$

Type Inference: Unification

Rule	Equations
$\Gamma, x : \alpha \vdash x : r$	$r \approx \alpha$
$\dfrac{\Gamma \vdash u : r_2 \quad \Gamma \vdash v : r_3}{\Gamma \vdash uv : r_1}$	$r_2 \approx r_3 \to r_1$
$\dfrac{\Gamma, x : r_2 \vdash u : r_3}{\Gamma \vdash \lambda x.u : r_1}$	$r_1 \approx r_2 \to r_3$

Definition [Unifier]

A *unifier* for equation system $\begin{cases} \alpha_1 \approx \beta_1 \\ \dots \\ \alpha_n \approx \beta_n \end{cases}$
is a substitution σ : $\quad \forall i.\ \sigma(\alpha_i) = \sigma(\beta_i)$

Example

$r_1 \to (r_2 \to r_3) \approx (r_4 \to r_5) \to r_6$
$r_1 := \alpha \to \alpha$
$r_6 := \alpha \to \alpha$
$r_{2,3,4,5} := \alpha$
$(\alpha \to \alpha) \to (\alpha \to \alpha)$

Definition [Most General Unifier; MGU]

MGU is a unifier σ_0 : $\forall \sigma'$—unifier $\exists \mu$ — substitution s.t. $\sigma' = \mu \circ \sigma_0$
(i.e. $\sigma'(\beta) = \mu(\sigma_0(\beta))$)

- MGU $\leftrightarrow$ principal type

Type Inference: Robinson’s algorithm

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn’t contain A iff

$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants ($p_1, \ldots, p_n$) and variables ($r_1, \ldots, r_n$)

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A <u>iff</u>
$\sigma = \sigma' \circ [r/A]$ <u>where</u> σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants ($p_1, \ldots, p_n$) and variables ($r_1, \ldots, r_n$)

1. $A \approx A$

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff $\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff
$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$(or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type

Type Inference: Robinson's algorithm

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff

$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants ($p_1, \ldots, p_n$) and variables ($r_1, \ldots, r_n$)

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$(or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type then fail
4. $r_j \approx B$(or $B \approx r_j$) where B contains but not equal to variable r_j

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff
$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$(or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type then fail
4. $r_j \approx B$(or $B \approx r_j$) where B contains but not equal to variable r_j then fail
5. $r_j \approx B$(or $B \approx r_j$) where B doesn't contain var r_j

Type Inference: Robinson's algorithm

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A <u>iff</u>

$\sigma = \sigma' \circ [r/A]$ <u>where</u> σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ <u>then</u> remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ <u>then</u> replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B(\text{or } B \approx p_i)$ <u>where</u> B — "constant", $\neq p_i$ or $\to$-type <u>then</u> fail
4. $r_j \approx B(\text{or } B \approx r_j)$ <u>where</u> B contains but not equal to variable r_j <u>then</u> fail
5. $r_j \approx B(\text{or } B \approx r_j)$ <u>where</u> B doesn't contain var r_j <u>then</u> recursive call with $\Sigma' := \Sigma[r_j/B]$

   <u>If</u> $\Sigma[r_j/B]$ is unifiable ($robinson(\Sigma[r_j/B]) = \sigma'_0$) <u>then</u> $\sigma_0 \overset{Lemma}{:=} \sigma'_0 \circ [r_j/B]$ unifies Σ

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff
$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$(or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type then fail
4. $r_j \approx B$(or $B \approx r_j$) where B contains but not equal to variable r_j then fail
5. $r_j \approx B$(or $B \approx r_j$) where B doesn't contain var r_j then recursive call with $\Sigma' := \Sigma[r_j/B]$

   If $\Sigma[r_j/B]$ is unifiable ($robinson(\Sigma[r_j/B]) = \sigma'_0$) then $\sigma_0 \overset{Lemma}{:=} \sigma'_0 \circ [r_j/B]$ unifies Σ
   More over σ_0 is MGU: for some unifier σ

   since σ'_0 is MGU for $\Sigma[r_j/B]$

$$\sigma \overset{Lemma}{=} \sigma' \circ [r_j/B] = \tau \circ \sigma'_0 \circ [r_j/B] \overset{def\ \sigma_0}{=} \tau \circ \sigma_0$$

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff
$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$ (or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type then fail
4. $r_j \approx B$ (or $B \approx r_j$) where B contains but not equal to variable r_j then fail
5. $r_j \approx B$ (or $B \approx r_j$) where B doesn't contain var r_j then recursive call with $\Sigma' := \Sigma[r_j/B]$

   If $\Sigma[r_j/B]$ is unifiable ($robinson(\Sigma[r_j/B]) = \sigma'_0$) then $\sigma_0 \overset{Lemma}{:=} \sigma'_0 \circ [r_j/B]$ unifies Σ
   More over σ_0 is MGU: for some unifier σ since σ'_0 is MGU for $\Sigma[r_j/B]$

$$\sigma \overset{Lemma}{=} \sigma' \circ [r_j/B] = \tau \circ \sigma'_0 \circ [r_j/B] \overset{def\ \sigma_0}{=} \tau \circ \sigma_0$$

- Why does it terminate?

Lemma

σ is a unifier for $\Sigma = r \approx A, \Sigma'$ and r doesn't contain A iff

$\sigma = \sigma' \circ [r/A]$ where σ' is a unifier for $\Sigma'[r/A]$

Robinson [MGU for Σ]

Type variables: constants $(p_1, \ldots, p_n)$ and variables $(r_1, \ldots, r_n)$

1. $A \approx A$ then remove
2. $A_1 \to A_2 \approx B_1 \to B_2$ then replace it with pair $A_1 \approx B_1, \quad A_2 \approx B_2$
3. $p_i \approx B$(or $B \approx p_i$) where B — "constant", $\neq p_i$ or $\to$-type then fail
4. $r_j \approx B$(or $B \approx r_j$) where B contains but not equal to variable r_j then fail
5. $r_j \approx B$(or $B \approx r_j$) where B doesn't contain var r_j then recursive call with $\Sigma' := \Sigma[r_j/B]$

   If $\Sigma[r_j/B]$ is unifiable ($robinson(\Sigma[r_j/B]) = \sigma'_0$) then $\sigma_0 \overset{Lemma}{:=} \sigma'_0 \circ [r_j/B]$ unifies Σ

   More over σ_0 is MGU: for some unifier σ

   since σ'_0 is MGU for $\Sigma[r_j/B]$

   $$\sigma \overset{Lemma}{=} \sigma' \circ [r_j/B] = \tau \circ \sigma'_0 \circ [r_j/B] \overset{def\ \sigma_0}{=} \tau \circ \sigma_0$$

- Why does it terminate?

Criteria	Number of vars	Number of $\to$	Number of $\approx$
Priority	0	1	2
Case	(5)	(2)	(1)

Domains

- base type $p \mapsto D_p$
- $D_{\sigma\to\tau} = (D_\sigma \to D_\tau) = D_\tau^{D_\sigma}$

Terms

- Vars: $\theta : x^\sigma \mapsto \theta_x \in D_\sigma$ — evaluation function
- Interpretation of terms: $[\![u]\!]_\theta \in D_\tau$ where τ is a type of u
  1. $[\![x]\!]_\theta = \theta(x)$
  2. $[\![uv]\!]_\theta = [\![u]\!]_\theta([\![v]\!]_\theta)$
  3. $[\![\lambda x.u]\!]_\theta = f : D_\sigma \to D_\tau$ where $x : \sigma, u : \tau$ $\quad f : a \mapsto [\![u]\!]_{\theta'}$ $\quad \theta'(y) = \begin{cases} \theta(y) & y \neq x \\ a & y = x \end{cases}$

STLC: Set-Theoretic Semantics

Domains

- base type $p \mapsto D_p$
- $D_{\sigma\to\tau} = (D_\sigma \to D_\tau) = D_\tau^{D_\sigma}$

Terms

- Vars: $\theta : x^\sigma \mapsto \theta_x \in D_\sigma$ — evaluation function
- Interpretation of terms: $[\![u]\!]_\theta \in D_\tau$ where τ is a type of u
  1. $[\![x]\!]_\theta = \theta(x)$
  2. $[\![uv]\!]_\theta = [\![u]\!]_\theta([\![v]\!]_\theta)$
  3. $[\![\lambda x.u]\!]_\theta = f : D_\sigma \to D_\tau$ where $x : \sigma, u : \tau$ $\quad f : a \mapsto [\![u]\!]_{\theta'}$ $\quad \theta'(y) = \begin{cases} \theta(y) & y \neq x \\ a & y = x \end{cases}$

Note:

$u : \tau \Rightarrow [\![u]\!]_\theta \in D_\tau$

Theorem [correctness]

$$u =_\beta v \Rightarrow \forall\theta.\ [\![u]\!]_\theta = [\![v]\!]_\theta$$

What about completeness?

$$\forall\theta.\ [\![u]\!]_\theta = [\![v]\!]_\theta \overset{?}{\Rightarrow} u =_\beta v$$

Domains

- base type $p \mapsto D_p$
- $D_{\sigma \to \tau} = (D_\sigma \to D_\tau) = D_\tau^{D_\sigma}$

Terms

- Vars: $\theta : x^\sigma \mapsto \theta_x \in D_\sigma$ — evaluation function
- Interpretation of terms: $[\![u]\!]_\theta \in D_\tau$ where τ is a type of u
  1. $[\![x]\!]_\theta = \theta(x)$
  2. $[\![uv]\!]_\theta = [\![u]\!]_\theta([\![v]\!]_\theta)$
  3. $[\![\lambda x.u]\!]_\theta = f : D_\sigma \to D_\tau$ where $x : \sigma, u : \tau$ $\quad f : a \mapsto [\![u]\!]_{\theta'} \quad \theta'(y) = \begin{cases} \theta(y) & y \neq x \\ a & y = x \end{cases}$

Note:

$u : \tau \Rightarrow [\![u]\!]_\theta \in D_\tau$

Theorem [correctness]

$u =_\beta v \Rightarrow \forall \theta.\ [\![u]\!]_\theta = [\![v]\!]_\theta$

What about completeness?

$\forall \theta.\ [\![u]\!]_\theta = [\![v]\!]_\theta \overset{?}{\Rightarrow} u =_\beta v$

completeness?: counterexample

$u = \lambda x^\alpha.(y^{\alpha \to \beta} x) : \alpha \to \beta$
$v = y$
$u \neq_\beta v$

$$[\![u]\!]_\theta = f : \alpha \mapsto [\![yx]\!]_{\theta[x/a]} \overset{def}{=} f : a \mapsto \underbrace{\theta(y)}_{\substack{\| \\ [\![y]\!]_{\theta[x/a]}}} (a) = [\![v]\!]_\theta$$

Definition [η-reduction]

$$\lambda x.(ux) \to_\eta u, \quad x \notin FV(u)$$

Properties $\to_{\beta\eta}$

- CR
- WN, SN
- Type preserving
- Interpretation preserving
- CR + WN $\Rightarrow \exists!\ \beta\eta$–NF

Definition [η-reduction]

$$\lambda x.(ux) \to_\eta u, \quad x \notin FV(u)$$

Properties $\to_{\beta\eta}$

- CR
- WN, SN
- Type preserving
- Interpretation preserving
- CR + WN $\Rightarrow \exists!\ \beta\eta$–NF

Lemma

$$u, v - \beta\eta - NF, \quad u \neq v \quad \Rightarrow \quad \exists\theta : \ [\![u]\!]_\theta \neq [\![v]\!]_\theta$$

Theorem [completeness]

$$(\forall D,\ \theta.\ [\![u]\!]_\theta = [\![v]\!]_\theta) \Rightarrow u =_{\beta\eta} v$$

Lemma [extensionality]

$$x\text{–fresh},\ ux =_{\beta\eta} vx \Rightarrow u =_{\beta\eta} v$$

STLC: Conclusion

- v Very simple
- v Type checking and type inference are decidable
- x Limited: no recursion!
- ⋗ Some function are typable not by type but by *type schemes*

  Example: $id \equiv \lambda x.x$ is typing by *type scheme* $\{\tau_1 \to \tau_1\}_{\tau_1}$ is a simple type

$$\tau ::= \alpha \mid \tau \rightarrow \tau \mid \forall \alpha . \tau$$

$$e ::= x \mid \lambda\, x \,.\, e \mid e_1\ e_2 \mid \Lambda\, \alpha \,.\, e \mid e\ \alpha$$

System F (aka $\lambda 2$)

$$\tau ::= \alpha \mid \tau \to \tau \mid \forall \alpha . \tau$$

$$e ::= x \mid \lambda\, x\, .\, e \mid e_1\ e_2 \mid \Lambda\, \alpha\, .\, e \mid e\ \alpha$$

Typing Rules

STLC $\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ [Var]$ $\dfrac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}\ [App]$ $\dfrac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}\ [Abs]$

$\dfrac{\Gamma \vdash e : \forall \alpha . \tau_2}{\Gamma \vdash e \tau_1 : \tau_2[\alpha / \tau_1]}\ [TApp]$ $\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash \Lambda \alpha . e : \forall \alpha . \tau}\ [TAbs]$

Examples

$3 \equiv \Lambda \alpha . \lambda x^{\alpha} . \lambda f^{\alpha \to \alpha} . f(f(f(x))) \ :\ \forall \alpha . \alpha \to (\alpha \to \alpha) \to \alpha$ $\qquad id \equiv \Lambda \alpha . \lambda x^{\alpha} . x \ :\ \forall \alpha . \alpha \to \alpha$

$$id(\forall \alpha . \alpha \to \alpha) \ :\ (\forall \alpha . \alpha \to \alpha) \to (\forall \alpha . \alpha \to \alpha)$$

What about Most General Type?

- Consider $\lambda x.xx$

$$\begin{array}{ll} x : \forall\alpha.\alpha \to \alpha & \vdash x : t \to t \\ x : \forall\alpha.\alpha \to \alpha & \vdash x : (t \to t) \to (t \to t) \\ x : \forall\alpha.\alpha \to \alpha & \vdash xx : (t \to t) \\ & \vdash \lambda x.xx : (\forall\alpha.\alpha \to \alpha) \to (\forall\beta.\beta \to \beta) \end{array}$$

- Is it the most general type?

What about Most General Type?

- Consider $\lambda x.xx$

$$
\begin{array}{ll}
x : \forall\alpha.\alpha \to \alpha & \vdash x : t \to t \\
x : \forall\alpha.\alpha \to \alpha & \vdash x : (t \to t) \to (t \to t) \\
x : \forall\alpha.\alpha \to \alpha & \vdash xx : (t \to t) \\
 & \vdash \lambda x.xx : (\forall\alpha.\alpha \to \alpha) \to (\forall\beta.\beta \to \beta)
\end{array}
$$

- Is it the most general type? **NO!**

$$\vdash \lambda x.xx : (\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)) \to (\forall\beta.(\beta \to \beta) \to (\beta \to \beta))$$

- Essence: $(\forall r.\xi) \to (\forall t.\eta)$
  - specification of η makes type more general
  - specification of ξ makes type more concrete
- Example: squaring function

$$
\begin{array}{rll}
dup = \lambda f.\lambda x.f\,(f\,x) & : & \forall\gamma.(\gamma \to \gamma) \to (\gamma \to \gamma) \\
\text{But not} & & \forall\rho.\rho \to \rho
\end{array}
$$

What about Most General Type?

- Consider $\lambda x.xx$

$$\begin{array}{ll} x : \forall\alpha.\alpha \to \alpha & \vdash x : t \to t \\ x : \forall\alpha.\alpha \to \alpha & \vdash x : (t \to t) \to (t \to t) \\ x : \forall\alpha.\alpha \to \alpha & \vdash xx : (t \to t) \\ & \vdash \lambda x.xx : (\forall\alpha.\alpha \to \alpha) \to (\forall\beta.\beta \to \beta) \end{array}$$

- Is it the most general type? **NO!**

$$\vdash \lambda x.xx : (\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)) \to (\forall\beta.(\beta \to \beta) \to (\beta \to \beta))$$

- Essence: $(\forall r.\xi) \to (\forall t.\eta)$
  - specification of η makes type more general
  - specification of ξ makes type more concrete
- Example: squaring function

$$\begin{array}{rl} dup = \lambda f.\lambda x.f\,(f\,x) \;\; : & \forall\gamma.(\gamma \to \gamma) \to (\gamma \to \gamma) \\ \text{But not} & \forall\rho.\rho \to \rho \end{array}$$

Type Annotations	Type Checking	Type Inference
Yes	Straightforward	Undecidable
No	Undecidable	Undecidable

Idea: Separate mono- from poly- morphic types
Also add type constructors

Terms

$$
\begin{array}{llll}
e & = & x & \text{Var} \\
 & \mid & e_1\ e_2 & \text{App} \\
 & \mid & \lambda\ x\ .\ e & \text{Abs} \\
 & \mid & \textbf{let}\ x = e_1\ \textbf{in}\ e_2 & \text{Let}
\end{array}
$$

Types

$$
\begin{array}{lllll}
\text{mono} & \tau & = & \alpha & \text{Type vars} \\
 & & \mid & C\ \tau_1 \ldots \tau_n & \text{Type Constructor} \\
\text{poly}^* & \sigma & = & \tau & \\
 & & \mid & \forall \alpha\ .\ \sigma &
\end{array}
$$

* aka *type schemes*

Examples

C = { $\texttt{Map}^2$, $\texttt{Int}^0$, $\texttt{->}^2$, ...}
Map String Int
Int -> Int
$\forall\alpha$. Map α Int -> [(α, Int)]

Hindley–Milner: Typing Rules

Typing Rules

Mono types!

STLC $\dfrac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$ [*Var*] $\dfrac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$ [*App*] $\dfrac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$ [*Abs*]

$\dfrac{\Gamma \vdash e : \sigma \quad \alpha \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash e : \forall \alpha.\sigma}$ [*Gen*] $\dfrac{\Gamma \vdash e_1 : \sigma \quad \Gamma, x : \sigma \vdash e_2 : \delta}{\Gamma \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \delta}$ [*Let*] $\dfrac{\Gamma \vdash e : \sigma' \quad \sigma' \sqsubseteq \sigma}{\Gamma \vdash e : \sigma}$ [*Inst*]

Type scheme

Typing Rules

Mono types!

STLC
$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}\ [Var] \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}\ [App] \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}\ [Abs]$$

$$\frac{\Gamma \vdash e : \sigma \quad \alpha \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash e : \forall \alpha.\sigma}\ [Gen] \qquad \frac{\Gamma \vdash e_1 : \sigma \quad \Gamma, x : \sigma \vdash e_2 : \delta}{\Gamma \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \delta}\ [Let] \qquad \frac{\Gamma \vdash e : \sigma' \quad \sigma' \sqsubseteq \sigma}{\Gamma \vdash e : \sigma}\ [Inst]$$

Type scheme

Example: let $id = \lambda x.x$ in $id : \forall \alpha.\alpha \to \alpha$

$$\cfrac{\cfrac{\cfrac{\cfrac{x : \alpha \in \{x : \alpha\}}{x : \alpha \vdash x : \alpha}\ [Var]}{\vdash \lambda x.x : \alpha \to \alpha}\ [Abs] \quad \alpha \notin \mathrm{FV}(\emptyset)}{\vdash \lambda x.x : \forall \alpha.\alpha \to \alpha}\ [Gen] \quad \cfrac{id : \forall \alpha.\alpha \to \alpha \in \{id : \forall \alpha.\alpha \to \alpha\}}{id : \forall \alpha.\alpha \to \alpha \vdash id : \forall \alpha.\alpha \to \alpha}\ [Var]}{\vdash \mathbf{let}\ id = \lambda x.x\ \mathbf{in}\ id : \forall \alpha.\alpha \to \alpha}\ [Let]$$

Alternative syntax:

(1)	$x : \alpha \vdash x : \alpha$	[Var]	$x : \alpha \in \{x : \alpha\}$
(2)	$\vdash \lambda x.x : \alpha \to \alpha$	[Abs]	(1)
(3)	$\vdash \lambda x.x : \forall \alpha.\alpha \to \alpha$	[Gen]	(2), $\alpha \notin \mathrm{FV}(\emptyset)$
(4)	$id : \forall \alpha.\alpha \to \alpha \vdash id : \forall \alpha.\alpha \to \alpha$	[Var]	$id : \forall \alpha.\alpha \to \alpha \in \{id : \forall \alpha.\alpha \to \alpha\}$
(5)	$\vdash \mathbf{let}\ id = \lambda x.x\ \mathbf{in}\ id : \forall \alpha.\alpha \to \alpha$	[Let]	(3), (4)

Instantiation

$$\frac{\Gamma \vdash e : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma \vdash e : \sigma} \quad \text{[Inst]}$$

Examples

$\forall \alpha.\ \alpha \to \alpha \sqsubseteq$ `Int -> Int`

$\forall \alpha.\ \alpha \to \alpha \sqsubseteq \forall \beta.\ \beta \to \beta$

$\forall \alpha\beta.\ \alpha \to \beta \to \alpha \sqsubseteq$ `Int -> Bool -> Int`

Instantiation (Specialization) formally

$\sigma \sqsubseteq \delta$ if $\exists$ substitution $S = \{\alpha_i/\tau_i\}$: $\qquad \sigma = S \circ \delta = \delta[S] = \delta[\{\alpha_i/\tau_i\}]$

$$\frac{\tau' = \tau[\{\alpha_i/\tau_i\}] \quad \beta_i \notin \mathrm{FV}(\forall \alpha_1 \ldots \forall \alpha_n.\tau)}{\forall \alpha_1 \ldots \forall \alpha_n.\tau \sqsubseteq \forall \beta_1 \ldots \forall \beta_m.\tau'}$$

- $\sqsubseteq$ is a partial order
- Principal type exists

$$\frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash (\textbf{let}\ x = e_1\ \textbf{in}\ e_2) : \tau} \quad \text{[Let]}$$

- Example:

```
let double f z = f (f z) in
  (double (λ x . x+1) 1, double (λ x . not x) false)
> :: (Int , Bool) = (3, false)
```

$$\frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash (\textbf{let}\ x = e_1\ \textbf{in}\ e_2) : \tau} \quad [\text{Let}]$$

- Example:

```
let double f z = f (f z) in
  (double (λ x . x+1) 1, double (λ x . not x) false)
> :: (Int , Bool) = (3, false)
```

- Note, the following is not typing in Hindley–Milner system:

```
(λ double . ( double (λ x . x+1)   1
            , double (λ x . not x) false) )
   (λ f z . f (f z))
```

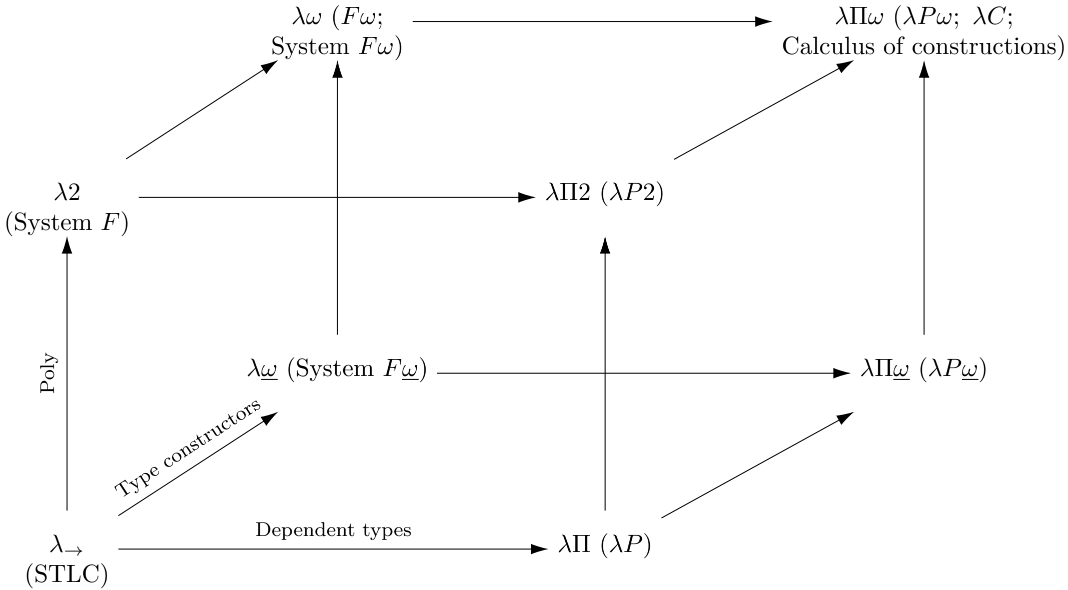
Hindley–Milner: To Take Away

- v Let-polymorphism
- v Type-inference is decidable (pretty the same way as in STLC)
- v Is a foundation for type systems in Haskell and ML
- ⋗ Extensions may break type inference decidability in Haskell:
  - GADT
  - RankNTypes ($\approx$ System F)
  - ...

Hindley–Milner: To Take Away

- v Let-polymorphism
- v Type-inference is decidable (pretty the same way as in STLC)
- v Is a foundation for type systems in Haskell and ML
- ⋗ Extensions may break type inference decidability in Haskell:
  - GADT
  - RankNTypes (≈ System F)
  - ...
- ⋗ Just for fun: try in GHCi! ☺

```
f a b c d = (a, b, c, d)
p1 = (f, f, f, f)
p2 = (p1, p1, p1, p1)
p3 = (p2, p2, p2, p2)
p4 = (p3, p3, p3, p3)
p5 = (p4, p4, p4, p4)
p6 = (p5, p5, p5, p5)
p7 = (p6, p6, p6, p6)
p8 = (p7, p7, p7, p7)
-- p9 = (p8, p8, p8, p8)
-- ...
ghci> :t p8
```


$\lambda\omega$ ($F\omega$;
System $F\omega$)
$\lambda\Pi\omega$ ($\lambda P\omega$; λC;
Calculus of constructions)
$\lambda 2$
(System F)
$\lambda\Pi 2$ ($\lambda P2$)
$\lambda\underline{\omega}$ (System $F\underline{\omega}$)
$\lambda\Pi\underline{\omega}$ ($\lambda P\underline{\omega}$)
Poly
Type constructors
Dependent types
$\lambda_{\rightarrow}$
(STLC)
$\lambda\Pi$ (λP)

Questions?