Introduction to Type Theory
February 2008
Alpha Lernet Summer School
Piriapolis, Uruguay

Herman Geuvers
Nijmegen & Eindhoven, NL

Lecture 2: Simple Type Theory: assigning types to untyped terms, principle types, normalization

Untyped λ-calculus

Simple Type Theory is not very expressive.

We can allow more functions to be definable by relaxing the type constraints. Most flexible : no types!

Untyped λ-calculus

$$\Lambda ::= \mathsf{Var} \mid (\Lambda\, \Lambda) \mid (\lambda \mathsf{Var}.\Lambda)$$

Examples:

- $\mathbf{K} := \lambda x\, y.x$
- $\mathbf{S} := \lambda x\, y\, z.x\, z(y\, z)$
- $\omega := \lambda x.x\, x$
- $\Omega := \omega\, \omega$

$$\Omega \longrightarrow_\beta \Omega$$

Untyped λ-calculus

Untyped λ-calculus is Turing complete

It's power lies in the fact that you can solve recursive equations:

Is there a term M such that

$$M\,x =_\beta x\,M\,x?$$

Is there a term M such that

$$M\,x =_\beta \mathbf{if}\,(\mathsf{Zero}\,x)\,\mathbf{then}\,1\,\mathbf{else}\,\mathsf{Mult}\,x\,(M\,(\mathsf{Pred}\,x))?$$

Yes, because we have a fixed point combinator.

Why do we want types?

- Types give a (partial) specification
- Typed terms can't go wrong (Milner) Subject Reduction property
- Typed terms always terminate
- The type checking algorithm detects (simple) mistakes

But:The compiler should compute the type information for us! (Why would the programmer hav eto type all that?)

This is called a type assignment system, or also typing à la Curry:

For M an untyped term, the type system assigns a type σ to M (or not)

STT à la Church and à la Curry

$\lambda\!\!\rightarrow$ (à la Church):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma\!\rightarrow\!\tau \;\; \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x{:}\sigma.P : \sigma\!\rightarrow\!\tau}$$

$\lambda\!\!\rightarrow$ (à la Curry):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma\!\rightarrow\!\tau \;\; \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x.P : \sigma\!\rightarrow\!\tau}$$

Exercise: Give a full derivation of

$$\vdash \lambda x.\lambda y.y(\lambda z.y\,x) : (\gamma\!\rightarrow\!\varepsilon)\!\rightarrow\!((\gamma\!\rightarrow\!\varepsilon)\!\rightarrow\!\varepsilon)\!\rightarrow\!\varepsilon$$

in Curry style $\lambda\!\!\rightarrow$

Examples

- Typed Terms:

$$\lambda x^{\alpha}.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^{\beta}.x))$$

  has only the type $\alpha\to((\beta\to\alpha)\to\alpha)\to\alpha$

- Type Assignment:

$$\lambda x.\lambda y.y(\lambda z.x))$$

  can be assigned the types

  - $\alpha\to((\beta\to\alpha)\to\alpha)\to\alpha$
  - $(\alpha\to\alpha)\to((\beta\to\alpha\to\alpha)\to\gamma)\to\gamma$
  - ...

  with $\alpha\to((\beta\to\alpha)\to\gamma)\to\gamma$ being the principal type

Connection between Church and Curry typed STT

Definition The erasure map $|-|$ from STT à la Church to STT à la Curry is defined by erasing all type information.

$$\begin{aligned} |x^{\alpha}| &:= x \\ |M\,N| &:= |M|\,|N| \\ |\lambda x^{\alpha}.M| &:= \lambda x.|M| \end{aligned}$$

So, e.g.

$$|\lambda x^{\alpha}.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^{\beta}.x))| = \lambda x.\lambda y.y(\lambda z.x))$$

Theorem If $M : \sigma$ in STT à la Church, then $|M| : \sigma$ in STT à la Curry.

Theorem If $P : \sigma$ in STT à la Curry, then there is an M such that $|M| \equiv P$ and $M : \sigma$ in STT à la Church.

Example of computing a principal type

$$\lambda x^{\alpha}.\lambda y^{\beta}.\underbrace{y^{\beta}(\lambda z^{\gamma}.\overbrace{y^{\beta}x^{\alpha}}^{\delta})}_{\varepsilon}$$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$.
2. Assign type vars to all applicative subterms: $y\,x : \delta$, $y(\lambda z.y\,x) : \varepsilon$.
3. Generate equations between types, necessary for the term to be typable: $\beta = \alpha{\to}\delta$ $\qquad\qquad$ $\beta = (\gamma{\to}\delta){\to}\varepsilon$
4. Find a most general unifier (a substitution) for the type vars that solves the equations: $\alpha := \gamma{\to}\delta$, $\beta := (\gamma{\to}\delta){\to}\varepsilon$, $\delta := \varepsilon$
5. The principal type of $\lambda x.\lambda y.y(\lambda z.yx)$ is now

$$(\gamma{\to}\varepsilon){\to}((\gamma{\to}\varepsilon){\to}\varepsilon){\to}\varepsilon$$

Exercise: Compute principal types for $\mathbf{S} := \lambda x.\lambda y.\lambda z.x\,z(y\,z)$ and for $M := \lambda x.\lambda y.x(y(\lambda z.x\,z\,z))(y(\lambda z.x\,z\,z))$.

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types. (Note: we can compose substitutions.)
- A unifier of the types σ and τ is a substitution that "makes σ and τ equal", i.e. an S such that $S(\sigma) = S(\tau)$
- A most general unifier (or mgu) of the types σ and τ is the "simplest substitution" that makes σ and τ equal, i.e. an S such that
  - $S(\sigma) = S(\tau)$
  - for all substitutions T such that $T(\sigma) = T(\tau)$ there is a substitution R such that $T = R \circ S$.

All these notions generalize to lists of types $\sigma_1, \ldots, \sigma_n$ in stead of pairs σ, τ.

Computability of most general unifiers

There is an algorithm U that, when given types $\sigma_1, \ldots, \sigma_n$ outputs

- A most general unifier of $\sigma_1, \ldots, \sigma_n$, if $\sigma_1, \ldots, \sigma_n$ can be unified.
- "Fail" if $\sigma_1, \ldots, \sigma_n$ can't be unified.

Definition σ is a principal type for the untyped λ-term M if

- $M : \sigma$ in STT à la Curry
- for all types τ, if $M : \tau$, then $\tau = S(\sigma)$ for some substitution S.

Theorem: Principal Types

There is an algorithm PT that, when given an (untyped) λ-term M, outputs

- A principal type σ such that $M : \sigma$ in STT à la Curry.
- "Fail" if M is not typable in STT à la Curry.

Typical problems one would like to have an algorithm for

$M : \sigma ?$	Type Checking Problem	TCP
$M : ?$	Type Synthesis or Type Assgnment Problem	TSP, TAP
$? : \sigma$	Type Inhabitation Problem (by a closed term)	TIP

For $\lambda\!\!\rightarrow$, all these problems are decidable,
both for the Curry style and for the Church style presentation.

Remarks:

- TCP and TSP are (usually) equivalent: To solve $MN : \sigma$, one has to solve $N :?$ (and if this gives answer τ, solve $M : \tau\!\rightarrow\!\sigma$).
- For Curry systems, TCP and TSP soon become undecidable if we go beyond $\lambda\!\!\rightarrow$.

- TIP is undecidable for most extensions of $\lambda\!\!\rightarrow$, as it corresponds to provability in some logic.

Properties of $\lambda\!\!\rightarrow$

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$.
- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$.
- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all $\beta\eta$-reductions from M terminate.
- Substitution property
  If $\Gamma, x : \tau, \Delta \vdash M : \sigma$, $\Gamma \vdash P : \tau$, then $\Gamma, \Delta \vdash M[P/x] : \sigma$.
- Thinning
  If $\Gamma \vdash M : \sigma$ and $\Gamma \subseteq \Delta$, then $\Delta \vdash M : \sigma$.

Normalization of β for $\lambda\!\!\rightarrow$

Note:

- Terms may get larger under reduction
  $(\lambda f.\lambda x.f(fx))P \longrightarrow_\beta \lambda x.P(Px)$
- Redexes may get multiplied under reduction.
  $(\lambda f.\lambda x.f(fx))((\lambda y.M)Q) \longrightarrow_\beta \lambda x.((\lambda y.M)Q)(((\lambda y.M)Q)x)$
- New redexes may be created under reduction.
  $(\lambda f.\lambda x.f(fx))(\lambda y.N) \longrightarrow_\beta \lambda x.(\lambda y.N)((\lambda y.N)x)$

First: Weak Normalization

- Weak Normalization: there is a reduction sequence that terminates,
- Strong Normalization: all reduction sequences terminate.

Towards Weak Normalization

There are three ways in which a "new" β-redex can be created.

- Creation

$$(\lambda x. \ldots x\, P \ldots)(\lambda y.Q) \longrightarrow_\beta \ldots (\lambda y.Q)P \ldots$$

- Multiplication

$$(\lambda x. \ldots x \ldots x \ldots)((\lambda y.Q)R) \longrightarrow_\beta \ldots (\lambda y.Q)R \ldots (\lambda y.Q)R \ldots$$

- Identity

$$(\lambda x.x)(\lambda y.Q)R \longrightarrow_\beta (\lambda y.Q)R$$

Towards Weak Normalization

Definition

The height (or order) of a type $h(\sigma)$ is defined by

- $h(\alpha) := 0$
- $h(\sigma_1 \to \ldots \to \sigma_n \to \alpha) := \mathsf{max}(h(\sigma_1), \ldots, h(\sigma_n)) + 1.$

NB [Exercise] This is the same as defining

- $h(\sigma \to \tau) := \mathsf{max}(h(\sigma) + 1, h(\tau)).$

Definition

The height of a redex $(\lambda x{:}\sigma.P)Q$ is the height of the type of $\lambda x{:}\sigma.P$

Towards Weak Normalization

Definition

We give a measure m to the terms by defining $m(N) := (h(N), \#N)$ with

- $h(N) =$ the maximum height of a redex in N,
- $\#N =$ the number of redexes of height $h(N)$ in N.

The measures of terms are ordered lexicographically:

$$(h_1, x) <_l (h_2, y) \text{ iff } h_1 < h_2 \text{ or } (h_1 = h_2 \text{ and } x < y)$$

.

Theorem: Weak Normalization

If P is a typable term in $\lambda\!\!\rightarrow$, then there is a terminating reduction starting from P.

Proof

Pick a redex of height $h(P)$ inside P that does not contain any other redex of height $h(P)$. [Note that this is always possible!]
Reduce this redex, to obtain Q. This does not create a new redex of height $h(P)$. [This is the important step. Exercise: check this; use the three ways in which new redexes can be created.]
So $m(Q) <_l m(P)$
As there are no infinitely decreasing $<_l$ sequences, this process must terminate and then we have arrived at a normal form.

Strong Normalization for $\lambda\!\!\rightarrow$ à la Curry

This is proved by constructing a model of $\lambda\!\!\rightarrow$.

Definition

- $[\![\alpha]\!] := \mathsf{SN}$ (the set of strongly normalizing λ-terms).
- $[\![\sigma\!\rightarrow\!\tau]\!] := \{M \mid \forall N \in [\![\sigma]\!](MN \in [\![\tau]\!])\}$.

Lemma

1. $xN_1 \ldots N_k \in [\![\sigma]\!]$ for all x, σ and $N_1, \ldots, N_k \in \mathsf{SN}$.
2. $[\![\sigma]\!] \subseteq \mathsf{SN}$
3. If $M[N/x]\vec{P} \in [\![\sigma]\!]$, $N \in \mathsf{SN}$, then $(\lambda x.M)N\vec{P} \in [\![\sigma]\!]$.

Strong Normalization for $\lambda\!\!\rightarrow$ à la Curry

Lemma

1. $xN_1 \ldots N_k \in [\![\sigma]\!]$ for all x, σ and $N_1, \ldots, N_k \in \mathsf{SN}$.
2. $[\![\sigma]\!] \subseteq \mathsf{SN}$
3. If $M[N/x]\vec{P} \in [\![\sigma]\!]$, $N \in \mathsf{SN}$, then $(\lambda x.M)N\vec{P} \in [\![\sigma]\!]$.

Proof: By induction on σ; the first two are proved simultaneously.
NB for the proof of (2): We need that $[\![\sigma]\!]$ is non-empty, which is guaranteed by the induction hypothesis for (1).
Also, use that $MN \in \mathsf{SN} \Rightarrow M \in \mathsf{SN}$. Think of it a bit and see it's true.

Proposition

$$\left.\begin{array}{l} x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash M : \sigma \\ N_1 \in [\![\tau_1]\!], \ldots, N_n \in [\![\tau_n]\!] \end{array}\right\} \Rightarrow M[N_1/x_1, \ldots N_n/x_n] \in [\![\sigma]\!]$$

Proof By induction on the derivation of $\Gamma \vdash M : \sigma$. (Using (3) of the previous Lemma.)

Corollary $\lambda\!\!\rightarrow$ is SN

Proof By taking $N_i := x_i$ in the Proposition. (That can be done, because $x_i \in [\![\tau_i]\!]$ by (1) of the Lemma.)
Then $M \in [\![\sigma]\!] \subseteq$ SN, using (2) of the Lemma. QED

Exercise Verify the details of the Strong Normalization proof. (That is, prove the Lemma and the Proposition.)

A little bit on semantics

$\lambda\!\!\rightarrow$ has a simple set-theoretic model. Given sets $[\![\alpha]\!]$ for type variables α, define

$$[\![\sigma\!\rightarrow\!\tau]\!] := [\![\tau]\!]^{[\![\sigma]\!]} \;\; (\text{ set theoretic function space } [\![\sigma]\!] \rightarrow [\![\tau]\!])$$

If any of the base sets $[\![\alpha]\!]$ is infinite, then there are higher and higher (uncountable) cardinalities among the $[\![\sigma]\!]$

There are smaller models, e.g.

$$[\![\sigma\!\rightarrow\!\tau]\!] := \{f \in [\![\sigma]\!] \rightarrow [\![\tau]\!] | f \text{ is definable}\}$$

where definability means that it can be constructed in some formal system. This restricts the collection to a countable set.

For example

$$[\![\sigma\!\rightarrow\!\tau]\!] := \{f \in [\![\sigma]\!] \rightarrow [\![\tau]\!] | f \text{ is } \lambda\text{-definable}\}$$