Type Theory and Coq

Herman Geuvers

Lecture: Simple Type Theory à la Curry: assigning types to untyped terms, principal type algorithm

Overview of todays lecture

- Simple Type Theory ($\lambda\!\rightarrow$) à la Curry (versus à la Church)
- Principal Types algorithm
- Properties of $\lambda\!\rightarrow$.
- Dependent Type Theory λP
- Type checking for λP.

Why do we want types? (programmers perspective)

- 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 have to 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)

$\lambda{\rightarrow}$ à la Church and à la Curry

$\lambda{\rightarrow}$ (à la Church):

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

$\lambda{\rightarrow}$ (à la Curry):

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

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

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

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

Connection between Church and Curry typed $\lambda\!\rightarrow$

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

$$\begin{array}{rcl} |x| & := & x \\ |M\,N| & := & |M|\,|N| \\ |\lambda x:\sigma.M| & := & \lambda x.|M| \end{array}$$

So, e.g.

$$|\lambda x:\alpha.\lambda y:(\beta\!\rightarrow\!\alpha)\!\rightarrow\!\alpha.y(\lambda z:\beta.x))| = \lambda x.\lambda y.y(\lambda z.x))$$

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

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

Connection between Church and Curry typed $\lambda\!\rightarrow$

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

$$\begin{array}{rcl} |x| & := & x \\ |M\,N| & := & |M|\,|N| \\ |\lambda x:\sigma.M| & := & \lambda x.|M| \end{array}$$

Theorem If $P:\sigma$ in $\lambda\!\rightarrow$ à la Curry, then there is an M such that $|M| \equiv P$ and $M:\sigma$ in $\lambda\!\rightarrow$ à la Church.
Proof: by induction on derivations.

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

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: $\beta = \alpha{\to}\delta$ $\quad\quad$ $\beta = (\gamma{\to}\delta){\to}\varepsilon$
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

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

Exercises to compute a principal type

1. Compute the principal type of $\mathbf{S} := \lambda x.\lambda y.\lambda z.x\, z(y\, z)$

2. Compute the principal type of
   $M := \lambda x.\lambda y.x(y(\lambda z.x\, z\, z))(y(\lambda z.x\, z\, z))$.

3. Consider the following two terms

   - $(\lambda x.\lambda y.x(\lambda z.y))\,(\lambda w.w)$
   - $(\lambda x.\lambda y.y(\lambda z.y))\,(\lambda w.w)$

   For each of these terms, compute its principle type, if it exists.
   Otherwise show that the principal type algorithm returns "reject".

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.

- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "reject" if $\alpha \in \mathsf{FV}(\tau_1)$, $\tau_1 \neq \alpha$.
- $U(\langle \sigma_1 = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \alpha = \sigma_1, \ldots, \sigma_n = \tau_n \rangle)$
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) := [\alpha := V(\tau_1), V]$, if $\alpha \notin \mathsf{FV}(\tau_1)$,
  where V abbreviates
  $U(\langle \sigma_2[\alpha := \tau_1] = \tau_2[\alpha := \tau_1], \ldots, \sigma_n[\alpha := \tau_1] = \tau_n[\alpha := \tau_1] \rangle)$.
- $U(\langle \mu \to \nu = \rho \to \xi, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \mu = \rho, \nu = \xi, \ldots, \sigma_n = \tau_n \rangle)$

Principal type

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

- $M : \sigma$ in $\lambda\!\rightarrow$ 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 $\lambda\!\rightarrow$ à la Curry.
- "Fail" if M is not typable in $\lambda\!\rightarrow$ à la Curry.

Typical problems one would like to have an algorithm for

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

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

- TCP and TSP are (usually) equivalent: To solve $MN : \sigma$, one has to solve $N :?$ (and if this gives answer τ, solve $M : \tau\to\sigma$).
- For Curry systems, TCP and TSP soon become undecidable beyond $\lambda\!\to$.
- TIP is undecidable for most extensions of $\lambda\!\to$, as it corresponds to provability in some logic.

λP: dependent type theory

Type checking is already difficult (interesting) for the Church case:

- types contain terms: "everything depends on everything"
- β-reduction inside types

λP-rules: axiom, application, abstraction, product

$$\frac{}{\vdash * : \Box}$$

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

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

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

λP-rules: weakening, variable, conversion

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \qquad \text{with } B =_\beta B'$$

Example

$$\begin{aligned}\Gamma \quad := \quad & A : *,\ \ c : A,\ \ R : A{\to}A{\to}*,\ \ f : A{\to}A, \\ & k : \Pi x{:}A.R\,c\,x, \\ & h : \Pi x, y{:}A.R\,x\,y{\to}R\,(f\,x)\,(f\,y), \\ & r : \Pi x, y{:}A.R\,x\,y{\to}R\,x\,(f\,y)\end{aligned}$$

Construct a term N such that

$$\Gamma \vdash N \quad : \quad \Pi x, y{:}A.R\,(f\,x)\,y{\to}R\,(f(f\,x))\,(f(f\,y)).$$

Curry-Howard-de Bruijn for minimal predicate logic

introduction rules versus abstraction rule

$$\dfrac{\begin{array}{c}[A^x] \\ \vdots \\ B\end{array}}{A \to B}\ I[x]{\to} \qquad \dfrac{\begin{array}{c}\vdots \\ B\end{array}}{\forall x.\, B}\ I\forall$$

$$\dfrac{\Gamma,\, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : s}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

elimination rules versus application rule

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ A \to B & A \end{array}}{B}\ E{\to} \qquad\qquad \dfrac{\begin{array}{c} \vdots \\ \forall x.\, B \end{array}}{B[x := N]}\ E\forall$$

$$\dfrac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

Example

Prove the following formula (and find an appropriate context to do so in)

$$(\forall x.\, P(x) \to Q(x)) \to (\forall x.\, P(x)) \to \forall y.\, Q(y)$$

Properties of λP

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma =_\beta \tau$.

- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash N : \sigma$.

- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all β-reductions from M terminate.

Proof of SN is by defining a reduction preserving map from λP to $\lambda{\rightarrow}$.

Decidability Questions

$$\begin{array}{ll} \Gamma \vdash M : \sigma? & \text{TCP} \\ \Gamma \vdash M : ? & \text{TSP} \\ \Gamma \vdash ? : \sigma & \text{TIP} \end{array}$$

For λP:

- TIP is undecidable
- TCP/TSP: simultaneously with Context checking

Type Checking

Define algorithms $\mathrm{Ok}(-)$ and $\mathrm{Type}_{\_}(-)$ simultaneously:

- $\mathrm{Ok}(-)$ takes a context and returns 'true' or 'false'
- $\mathrm{Type}_{\_}(-)$ takes a context and a term and returns a term or 'false'.

The type synthesis algorithm $\mathrm{Type}_{\_}(-)$ is sound if

$$\mathrm{Type}_{\Gamma}(M) = A \;\;\Rightarrow\;\; \Gamma \vdash M : A$$

for all Γ and M.

The type synthesis algorithm $\mathrm{Type}_{\_}(-)$ is complete if

$$\Gamma \vdash M : A \;\;\Rightarrow\;\; \mathrm{Type}_{\Gamma}(M) =_{\beta} A$$

for all Γ, M and A.

$$
\begin{array}{rcl}
\mathrm{Ok}(<>) & = & \text{‘true’} \\[2em]
\mathrm{Ok}(\Gamma, x{:}A) & = & \mathrm{Type}_\Gamma(A) \in \{*, \mathbf{kind}\}, \\[2em]
\mathrm{Type}_\Gamma(x) & = & \text{if } \mathrm{Ok}(\Gamma) \text{ and } x{:}A \in \Gamma \text{ then } A \text{ else ‘false’}, \\[2em]
\mathrm{Type}_\Gamma(\mathbf{type}) & = & \text{if } \mathrm{Ok}(\Gamma) \text{then } \mathbf{kind} \text{ else ‘false’}, \\[2em]
\mathrm{Type}_\Gamma(MN) & = & \text{if } \mathrm{Type}_\Gamma(M) = C \text{ and } \mathrm{Type}_\Gamma(N) = D \\
& & \quad \text{then} \quad \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\
& & \qquad\qquad \text{then } B[x := N] \text{ else ‘false’} \\
& & \quad \text{else} \quad \text{‘false’},
\end{array}
$$

$$
\begin{aligned}
\mathrm{Type}_\Gamma(\lambda x{:}A.M) \quad &= \quad \text{if } \mathrm{Type}_{\Gamma,x:A}(M) = B \\
&\quad\ \text{then} \qquad\qquad \text{if } \mathrm{Type}_\Gamma(\Pi x{:}A.B) \in \{\mathbf{type}, \mathbf{kind}\} \\
&\qquad\qquad\qquad\qquad \text{then } \Pi x{:}A.B \text{ else 'false'} \\
&\quad\ \text{else 'false'}, \\
\mathrm{Type}_\Gamma(\Pi x{:}A.B) \quad &= \quad \text{if } \mathrm{Type}_\Gamma(A) = \mathbf{type} \text{ and } \mathrm{Type}_{\Gamma,x:A}(B) = s \\
&\quad \text{then } s \text{ else 'false'}
\end{aligned}
$$

Soundness and Completeness

Soundness

$$\text{Type}_\Gamma(M) = A \;\Rightarrow\; \Gamma \vdash M : A$$

Completeness

$$\Gamma \vdash M : A \;\Rightarrow\; \text{Type}_\Gamma(M) =_\beta A$$

As a consequence:

$$\text{Type}_\Gamma(M) = \text{'false'} \;\Rightarrow\; M \text{ is not typable in } \Gamma$$

NB 1. Completeness implies that Type terminates on all well-typed terms. We want that Type terminates on all pseudo terms.

NB 2. Completeness only makes sense if we have uniqueness of types (Otherwise: let $\text{Type}_{\_}(-)$ generate a set of possible types)

Termination

We want $\mathrm{Type}_{\_}(-)$ to terminate on all inputs.

Interesting cases: λ-abstraction and application:

$$\begin{array}{lcl} \mathrm{Type}_\Gamma(\lambda x{:}A.M) & = & \text{if } \mathrm{Type}_{\Gamma,x:A}(M) = B \\ & & \text{then} \qquad\qquad \text{if } \mathrm{Type}_\Gamma(\Pi x{:}A.B) \in \{\mathbf{type}, \mathbf{kind}\} \\ & & \qquad\qquad\qquad\quad \text{then } \Pi x{:}A.B \text{ else 'false'} \\ & & \text{else 'false'}, \end{array}$$

! Recursive call is not on a smaller term!

Replace the side condition

$$\text{if } \mathrm{Type}_\Gamma(\Pi x{:}A.B) \in \{\mathbf{type}, \mathbf{kind}\}$$

by

$$\text{if } \mathrm{Type}_\Gamma(A) \in \{\mathbf{type}\}$$

Termination

We want $\mathrm{Type}_{\_}(-)$ to terminate on all inputs.

Interesting cases: λ-abstraction and application:

$$\begin{array}{lcll} \mathrm{Type}_\Gamma(MN) & = & \text{if } \mathrm{Type}_\Gamma(M) = C \text{ and } \mathrm{Type}_\Gamma(N) = D & \\ & & \text{then} & \text{if } C \twoheadrightarrow_\beta \Pi x{:}A.B \text{ and } A =_\beta D \\ & & & \text{then } B[x := N] \text{ else 'false'} \\ & & \text{else} & \text{'false'}, \end{array}$$

! Need to decide β-reduction and β-equality!

For this case, termination follows from soundness of Type and the decidability of equality on well-typed terms (using SN and CR).