Lambda-Calculus and Type Theory

ISR 2024
Obergurgl, Austria

Herman Geuvers & Niels van der Weide

Radboud University Nijmegen, The Netherlands

Lecture 12

Principal types: a functional programmers' view on type theory

Why do programmers want types?

- Types give a (partial) specification
- Typed terms can't go wrong (Milner)
  Subject Reduction property: If $M : A$ and $M \twoheadrightarrow_\beta N$, then $N : A$.
- 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)

Simple Type Theory 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):

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

Type Assignment systems

- With typed assignment also called typing à la Curry, we assign types to untyped λ-terms

$$\lambda x.x : \alpha \to \alpha$$

- As a consequence:
  - Terms do not have unique types,
  - A principal type can be computed using unification.
- Example:

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

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 $\lambda\!\!\rightarrow$

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

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

So, e.g. $\quad |\lambda x:\alpha.\lambda y:(\beta\!\rightarrow\!\alpha)\!\rightarrow\!\alpha.y(\lambda z:\beta.x)| = \lambda x.\lambda y.y(\lambda z.x)$

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

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

Example of computing a principal type

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

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

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

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

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

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

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

Example of computing a principal type (II)

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

Which of these terms is typable?

- $M_1 := \lambda x.x\,(\lambda y.y\,x)$
- $M_2 := \lambda x.\lambda y.x\,(x\,y)$
- $M_3 := \lambda x.\lambda y.x\,(\lambda z.y\,x)$

Poll:

A M_1 is not typable, M_2 and M_3 are typable.

B M_2 is not typable, M_1 and M_3 are typable.

C M_3 is not typable, M_1 and M_2 are typable.

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types with a finite domain such that variables that occur in the range of S are not in the domain of S.
- A substitution S is written as $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$ where
  - all α_i are different
  - $\alpha_i \notin \sigma_j$ (for all i, j).
- We write $\tau\, S$ for substitution S applied to τ.
- We can compose substitutions: $S; T$. (So we have $\tau\,(S; T) = (\tau\, S)\, T$.)
- A unifier of the types σ and τ is a substitution that solves $\sigma = \tau$, i.e. an S such that $\sigma\, S = \tau\, S$
- A most general unifier (mgu) of the types σ and τ is the "simplest substitution" that solves $\sigma = \tau$, i.e. an S such that
  - $\sigma\, S = \tau\, S$
  - for all substitutions T such that $\sigma\, T = \tau\, T$ there is a substitution R such that $T = S; R$.

Principal Types: solving a list of equations

All notions generalize to lists of equations

$$\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$$

instead of a single equation $\sigma = \tau$.

- A unifier of $\mathcal{E}$ is a substitution S such that $\sigma_i\, S = \tau_i\, S$ for all i.
- A most general unifier (mgu) for $\mathcal{E}$ is an S such that
  - $\sigma_i\, S = \tau_i\, S$ for all i
  - for all substitutions T such that $\sigma_i\, T = \tau_i\, T$ for all i, there is a substitution R such that $T = S; R$.

Computability of most general unifiers

THEOREM There is an algorithm U that, given a list of equations $\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\mathcal{E}$ if these equations can be solved.
- "Fail" if $\mathcal{E}$ can't be solved.

PROOF

- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "Fail" 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 closed λ-term M if

- $\vdash M : \sigma$ in $\lambda\!\!\rightarrow$ à la Curry
- for all types τ, if $\vdash M : \tau$, then $\tau = \sigma\, T$ for some substitution T.

Principal Type Theorem

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

- A principal type σ for M if M is typable in $\lambda\!\!\rightarrow$ à la Curry.
- "Fail" if M is not typable in $\lambda\!\!\rightarrow$ à la Curry.

PROOF In the algorithm we

- first label the bound variables and all applicative sub-terms with type variables, and we give the candidate type τ,
- then we generate the equations that need to hold for the term to be typable,
- then we compute the mgu of this set of equations and we obtain the substitution S or "Fail",
- then we have as output the principal type $\tau\, S$ or "Fail".

The proof that this output indeed correctly computes the principal type can be found in the literature.

Principal Types for open terms

The definitions and the theory for principal types immediately extends open terms.

Consider for example the term

$$M := y\,(\lambda z.y\,x)$$

We are looking for a principal pair $(x : \sigma_1, y : \sigma_2; \sigma)$ satisfying

- $x : \sigma_1, y : \sigma_2 \vdash M : \sigma$,
- for all τ_1, τ_2, τ satisfying $x : \tau_1, y : \tau_2 \vdash M : \tau$ there is a substitution T such that $\tau_1\,T = \sigma_1$, $\tau_2\,T = \sigma_2$ and $\tau\,T = \sigma$.

We do this by closing M under λ-abstractions obtaining

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

for which we have the principal type

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

And so the principal pair for M is

$$(x : \gamma{\to}\varepsilon, y : (\gamma{\to}\varepsilon){\to}\varepsilon; \varepsilon).$$

Typical problems one would like to have an algorithm for

$\Gamma \vdash M : \sigma ?$	Type Checking Problem	TCP
$\Gamma \vdash M : ?$	Type Synthesis Problem	TSP
$\Gamma \vdash ? : \sigma$	Type Inhabitation Problem	TIP

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