Proving with Computer Assistance

Herman Geuvers

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

Overview of todays lecture

- Recap of Simple Type Theory a la Church
- Recap of Untyped lambda calculus
- Simple Type Theory a la Curry (versus a la Church)
  A programmers view on type theory
- Principal Types algorithm
- Properties of Simple Type Theory.

Recap: Simple type theory a la Church.

Formulation with contexts to declare the free variables:

$$x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n$$

is a context, usually denoted by Γ.
Derivation rules of $\lambda{\to}$ (à la Church):

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

$\Gamma \vdash_{\lambda\to} M : \sigma$ if there is a derivation using these rules with conclusion $\Gamma \vdash M : \sigma$

Recap: Formulas-as-Types (Curry, Howard)

There are two readings of a judgement $M : \sigma$

1. term as algorithm/program, type as specification:
   M is a function of type σ
2. type as a proposition, term as its proof:
   M is a proof of the proposition σ

- There is a one-to-one correspondence:

  typable terms in $\lambda\!\!\rightarrow$ $\simeq$ derivations in minimal proposition logic

- $x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n \vdash M : \sigma$ can be read as
  M is a proof of σ from the assumptions $\tau_1, \tau_2, \ldots, \tau_n$.

Recap: Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\alpha\to\beta\to\gamma]^3 \quad [\alpha]^1}{\beta\to\gamma} \qquad \dfrac{[\alpha\to\beta]^2 \quad [\alpha]^1}{\beta}}{\gamma}}{\alpha\to\gamma}\,1}{(\alpha\to\beta)\to\alpha\to\gamma}\,2}{(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma}\,3$$

$$\simeq$$

$$\lambda x{:}\alpha\to\beta\to\gamma.\lambda y{:}\alpha\to\beta.\lambda z{:}\alpha.xz(yz)$$
$$: (\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma$$

Example

Find a term M of type $((A \to B) \to A) \to (A \to A \to B) \to A$, and give a typing derivation that shows this typing.

Untyped λ-calculus

Untyped λ-calculus

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

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

$$\Omega \to_\beta \Omega$$

Untyped λ-calculus

Untyped λ-calculus is Turing complete
Its 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

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

Yes, because we have a fixed point combinator:
- $\mathbf{Y} := \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$

Property:

$$\mathbf{Y}\,f =_\beta f(\mathbf{Y}\,f)$$

Untyped λ-calculus (ctd.)

Solving recursive equations using the fixed point combinator:

- For M a λ-term, $\mathbf{Y}\,M$ is a fixed point of M, that is

$$M\,(\mathbf{Y}\,M) =_\beta \mathbf{Y}\,M$$

- As a consequence, a question like "Is there a λ-term P such that $P\,x =_\beta x\,P\,x\,P$ (for all x)?" can be answered affirmative:

Why do we 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

$\lambda\!\to$ (à la Church):

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

$\lambda\!\to$ (à la Curry):

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

Typed Terms versus Type Assignment:

- With typed terms also called typing à la Church, we have terms with type information in the λ-abstraction

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

  As a consequence:

  - Terms have unique types,
  - The type is directly computed from the type info in the variables.

- 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.

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

Example derivation

$\lambda x.\lambda y.y(\lambda z.x)$ can be assigned the type
$(\alpha\to\alpha)\to((\beta\to\alpha\to\alpha)\to\gamma)\to\gamma$ in $\lambda\to$ a la Curry.

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.

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

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.

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

Definition The erasure map $|-|$ from $\lambda\!\!\to$ à la Church to $\lambda\!\!\to$ 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 $\Gamma \vdash P : \sigma$ in $\lambda\!\!\to$ à la Curry, then there is an M such that $|M| \equiv P$ and $\Gamma \vdash M : \sigma$ in $\lambda\!\!\to$ à la Church.
Proof: by induction on derivations.

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

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

4. Find a most general unifier (a substitution) for the type vars that solves the equations: $\alpha := \gamma \to \varepsilon,\ \beta := (\gamma \to \varepsilon) \to \varepsilon,\ \delta := \varepsilon$

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)

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

Steps in computing the most general unifier

$$\lambda x.\lambda y.x\,y\,x$$

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 and such that variables that occur in the range of S are not in the domain of S.
- We write S as $[\alpha_1 := \sigma_1, \dots, \alpha_n := \sigma_n]$ with $\alpha_i \notin \sigma_j$ $(\forall i, j)$.
- We can compose substitutions: $S; T$. We write $\tau\, S$ for substitution S applied to τ. (So we have $\tau\,(S; T) = (\tau\, S)\, T$.)
- A unifier of the types σ and τ is a substitution that "makes $\sigma = \tau$ hold, i.e. an S such that $\sigma\, S = \tau\, S$
- A most general unifier (or mgu) of the types σ and τ is the "simplest substitution" that makes $\sigma = \tau$ hold, 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 $T = S; R$.

All these notions generalize to lists of equations $\langle \sigma_1 = \tau_1, \dots, \sigma_n = \tau_n \rangle$ instead of a single equation $\sigma = \tau$.

Computability of most general unifiers

There is an algorithm U that, given a list $\langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ if these types can be unified.
- "Fail" if $\langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ can't be unified.
- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "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 $\tau = \sigma\, S$ for some substitution S.

Theorem: Principal Types

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

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

NB. The definitions and theory can be immediately extended to deal with open terms.

Typical problems one would like to have an algorithm for

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

For $\lambda\!\!\rightarrow$, all these problems are decidable,
both for the Curry style and for the Church style presentation
(also if we ask them in a context Γ).

- TCP and TSP are (as usual) 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 beyond $\lambda\!\!\rightarrow$.
- TIP is undecidable for most extensions of $\lambda\!\!\rightarrow$, as it corresponds to provability in some logic.