Introduction to the simply-typed λ-calculus

Frédéric Blanqui

INRIA

2nd Asian-Pacific Summer School on Formal Methods
Tsinghua University, 20-27 August 2010

Outline

Using the untyped λ-calculus as a programming language ?

This is possible! cf. J.-J. Lévy's lecture.

Examples: LISP (1958), Scheme (1975), ...

Problem: what to do with expressions like

$$(\lambda x.\texttt{if } x \geq 2 \texttt{ then } t \texttt{ else } u)\text{"}\texttt{foo}\text{"} ?$$

Typed programming languages like Pascal (1970), C (1972), ML (1973), Ada (1977), C++ (1979), Coq (1985), Java (1995), OCaml (1996), ... reject such ill-typed expressions.

Simple types

A simple type is either:

- a type constant $\mathrm{bool}, \mathrm{int}, \mathrm{float}, \ldots \in \mathcal{B}$
- a function type $S \rightarrow T$, where S and T are themselves types

Remark: every type is of the form

$$T_1 \rightarrow \ldots \rightarrow T_n \rightarrow B$$

where $n \geq 0$ and $B \in \mathcal{B}$.

Order of a type

$$order(T_1 \to \ldots \to T_n \to B) = max(\{0\} \cup \{1 + order(T_i) | 1 \leq i \leq n\})$$

Examples:

- $order(\mathrm{int}) = 0$
- $order(\mathrm{int} \to \mathrm{int}) = order(\mathrm{int} \to \mathrm{int} \to \mathrm{int}) = 1$
- $order((\mathrm{int} \to \mathrm{int}) \to \mathrm{int}) = 2$

Most programming languages allow types of order 1 only...
ML, OCaml, Coq, ... allow types of any order.
Coq allows even richer types (polymorphic, dependent, ...)

Assigning a type to a λ-term

Problem: what type(s) has $\lambda x.x$?

$\mathrm{bool} \rightarrow \mathrm{bool}$, $\mathrm{int} \rightarrow \mathrm{int}$, $(\mathrm{int} \rightarrow \mathrm{int}) \rightarrow (\mathrm{int} \rightarrow \mathrm{int})$,...
are all possible

$\Rightarrow$ a typable expression has no unique type !

Types of variables ?

Two approaches:

- *à la Curry* (1934): variables are not annotated

  

  $\lambda x.t$

  $\Rightarrow$ a typable expression has no unique type
  BUT has a unique most general type schema (proof later)

- *à la Church* (1940): variables are annotated with their type

  

  $\lambda x^S.t$

  $\Rightarrow$ a typable expression has a unique type (proof later)

Typing environments

Notations:

- $\mathcal{X}$ is the set of variables $x, y, \ldots$
- $\mathcal{L}$ is the set of λ-terms $s, t, \ldots$
- $\mathcal{T}$ is the set of simple types $S, T, \ldots$
- $\mathcal{E}$ is the set of typing environments $\Gamma, \Delta, \ldots$
  i.e. the set of finite maps from $\mathcal{X}$ to $\mathcal{T}$

$\mathrm{dom}(\Gamma) = \{x \in \mathcal{X} \mid \exists T, (x, T) \in \Gamma\}$ is the domain of Γ

Assigning a type to a λ-term - Church approach

Typing $\vdash$ is inductively defined as the smallest relation on $\mathcal{E} \times \mathcal{L} \times \mathcal{T}$ such that:

(var) $(\Gamma, x, S) \in \vdash$
if $(x, S) \in \Gamma$

(abs) $(\Gamma, \lambda x^S.t, S \rightarrow T) \in \vdash$
if $x \notin \mathrm{dom}(\Gamma)$ and $(\Gamma \cup \{(x, S)\}, t, T) \in \vdash$

(app) $(\Gamma, us, T) \in \vdash$
if there is $S \in \mathcal{T}$ such that $(\Gamma, u, S \rightarrow T) \in \vdash$ and $(\Gamma, s, S) \in \vdash$

Assigning a type to a λ-term - Church approach

By writing $\Gamma \vdash v : V$ instead of $(\Gamma, v, V) \in \vdash$ and using deduction rules...

Typing $\vdash$ is inductively defined as the smallest relation on $\mathcal{E} \times \mathcal{L} \times \mathcal{T}$ such that:

(var) $\dfrac{}{\Gamma \vdash x : S}$ if $(x, S) \in \Gamma$

(abs) $\dfrac{\Gamma \cup \{(x, S)\} \vdash t : T}{\Gamma \vdash \lambda x^S.t : S \to T}$ if $x \notin \mathrm{dom}(\Gamma)$

(app) $\dfrac{\Gamma \vdash u : S \to T \quad \Gamma \vdash s : S}{\Gamma \vdash us : T}$

Example

Let $\Gamma = \{(\leq, \mathrm{int} \to \mathrm{int} \to \mathrm{bool}), (2, \mathrm{int}), (x, \mathrm{int}), (t, \mathrm{int}), (u, \mathrm{int}),$
$(\texttt{if \_ then \_ else \_}, \mathrm{bool} \to \mathrm{int} \to \mathrm{int} \to \mathrm{int})\}.$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \_ \leq \_ : \mathrm{int} \to \mathrm{int} \to \mathrm{bool}}\,(\text{var}) \qquad \dfrac{}{\Gamma \vdash x : \mathrm{int}}\,(\text{var})}{\Gamma \vdash x \leq \_ : \mathrm{int} \to \mathrm{bool}}\,(\text{app}) \qquad \dfrac{}{\Gamma \vdash 2 : \mathrm{int}}\,(\text{var})}{\Gamma \vdash x \leq 2 : \mathrm{bool}}\,(\text{app})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \texttt{if \_ then \_ else \_} : \mathrm{bool} \to \mathrm{int} \to \mathrm{int} \to \mathrm{int}}\,(\text{var}) \qquad \dfrac{\cdots}{\Gamma \vdash x \leq 2 : \mathrm{bool}}}{\Gamma \vdash \texttt{if } x \leq 2 \texttt{ then \_ else \_} : \mathrm{int} \to \mathrm{int} \to \mathrm{int}}\,(\text{ap} \\ \cdots}{\Gamma \vdash \texttt{if } x \leq 2 \texttt{ then } t \texttt{ else } u : \mathrm{int}}}{\vdash \lambda x.\texttt{if } x \leq 2 \texttt{ then } t \texttt{ else } u : \mathrm{int} \to \mathrm{int}}\,(\text{abs})$$

Assigning a type to a λ-term - Curry approach

Typing $\vdash$ is inductively defined as the smallest relation on $\mathcal{E} \times \mathcal{L} \times \mathcal{T}$ such that:

- $(\Gamma, \lambda x.t, S \rightarrow T) \in \vdash$
  if $x \notin \mathrm{dom}(\Gamma)$ and $(\Gamma \cup \{(x, S)\}, t, T) \in \vdash$

$$\text{(abs)}\ \frac{\Gamma \cup \{(x, S)\} \vdash t : T}{\Gamma \vdash \lambda x.t : S \rightarrow T} \text{ if } x \notin \mathrm{dom}(\Gamma)$$

Outline

Decidability of type-checking

Problem: given Γ, v and V, is it decidable whether $\Gamma \vdash v : V$?

Inversion lemma

Lemma: assume that $\Gamma \vdash v : V$.

- If $v \in \mathcal{X}$,
  then $(v, V) \in \Gamma$.
- If $v = \lambda x^S.t$ and $x \notin \mathrm{dom}(\Gamma)$,
  then there is T such that $V = S \to T$ and $\Gamma \cup \{(x, S)\} \vdash t : T$.
- If $v = us$,
  then there is S such that $\Gamma \vdash u : S \to V$ and $\Gamma \vdash s : S$.

Unicity of type in Church approach

Theorem: if $\Gamma \vdash v : V$ and $\Gamma \vdash v : V'$, then $V = V'$.

Proof. By induction on $\Gamma \vdash v : V$.

(var) $\dfrac{}{\Gamma \vdash x : S}$ if $(x, S) \in \Gamma$. We have $v = x$ and $V = S$. By inversion, $(x, V') \in \Gamma$. Since Γ is a function, $V = V'$.

(abs) $\dfrac{\Gamma \cup \{(x, S)\} \vdash t : T}{\Gamma \vdash \lambda x^S.t : S \to T}$ if $x \notin \mathrm{dom}(\Gamma)$. We have $v = \lambda x^S.t$ and $V = S \to T$. By inversion, there is T' such that $V' = S \to T'$ and $\Gamma \cup \{(x, S)\} \vdash t : T'$. By IH, $T = T'$. Thus, $V = V'$.

(app) $\dfrac{\Gamma \vdash u : S \to T \quad \Gamma \vdash s : S}{\Gamma \vdash us : T}$. We have $v = us$ and $V = T$. By inversion, there is $S' \in \mathcal{T}$ such that $\Gamma \vdash u : S' \to V'$ and $\Gamma \vdash s : S'$. By IH, $S \to V = S' \to V'$. Thus, $V = V'$.

Decidability of type-checking

Problem: given Γ, v and V, is it decidable whether $\Gamma \vdash v : V$?

We first define a computable function ϕ which, given Γ and v, returns the unique type of v in Γ if it exists, and error otherwise.

- $\phi(\Gamma, x) = S$ if $(x, S) \in \Gamma$
- $\phi(\Gamma, \lambda x^S.t) = S \to T$ if $\phi(\Gamma \cup \{(x, S)\}, t) = T$
- $\phi(\Gamma, us) = T$ if $\phi(\Gamma, u) = S \to T$ and $\phi(\Gamma, s) = S$
- $\phi(\Gamma, v) = \mathrm{error}$ otherwise

Then, the following function answers the problem:

- $\psi(\Gamma, v, V) = \mathrm{true}$ if $\phi(\Gamma, v) = V$
- $\psi(\Gamma, v, V) = \mathrm{false}$ otherwise

Correctness

Lemma: if $\phi(\Gamma, v) = V \neq \mathrm{error}$, then $\Gamma \vdash v : V$.

Proof. By induction on v.

- $v \in \mathcal{X}$. By assumption, $(v, V) \in \Gamma$. Thus, $\Gamma \vdash v : V$.
- $v = \lambda x^S.t$. By assumption, $\phi(\Gamma \cup \{(x, S)\}, t) = T \neq \mathrm{error}$ and $V = S \to T$. By IH, $\Gamma \cup \{(x, S)\} \vdash t : T$. Thus, $\Gamma \vdash v : V$.
- $v = us$. By assumption, there is S such that $\phi(\Gamma, u) = S \to T$ and $\phi(\Gamma, s) = S$. By IH, $\Gamma \vdash u : S \to T$ and $\Gamma \vdash s : S$. Thus, $\Gamma \vdash v : V$.

Corollary: if $\psi(\Gamma, v, V) = \mathrm{true}$, then $\Gamma \vdash v : V$.

Completeness

Lemma: if $\Gamma \vdash v : V$, then $\phi(\Gamma, v) = V \neq \mathrm{error}$.

Proof. By induction on $\Gamma \vdash v : V$.

(var) $\dfrac{}{\Gamma \vdash x : S}$ if $(x, S) \in \Gamma$. We have $v = x$ and $V = S$. Thus, $\phi(\Gamma, v) = V$.

(abs) $\dfrac{\Gamma \cup \{(x, S)\} \vdash t : T}{\Gamma \vdash \lambda x^S.t : S \to T}$ if $x \notin \mathrm{dom}(\Gamma)$. We have $v = \lambda x^S.t$ and $V = S \to T$. By IH, $\phi(\Gamma \cup \{(x, S)\}, t) = T \neq \mathrm{error}$. Thus, $\phi(\Gamma, v) = V \neq \mathrm{error}$.

(app) $\dfrac{\Gamma \vdash u : S \to T \quad \Gamma \vdash s : S}{\Gamma \vdash us : T}$. We have $v = us$ and $V = T$. By IH, $\phi(\Gamma, u) = S \to T \neq \mathrm{error}$ and $\phi(\Gamma, s) = S \neq \mathrm{error}$. Thus, $\phi(\Gamma, v) = V \neq \mathrm{error}$.

Corollary: if $\Gamma \vdash v : V$, then $\psi(\Gamma, v, V) = \mathrm{true}$.

Outline

Type inference

Problem: given v, is it decidable whether there exists Γ and V such that $\Gamma \vdash v : V$?

Problem for completeness: v has no unique type...

$$\lambda x.x \text{ has type } \mathrm{int} \to \mathrm{int},\ (\mathrm{int} \to \mathrm{int}) \to (\mathrm{int} \to \mathrm{int}),\ \ldots$$

Idea: use type variables !

every type of $\lambda x.x$ is an instance of the type schema $\alpha \to \alpha$

Type schema

Types and typing are extended with type variables:

- $\mathcal{S}$ is the set of type schema $S, T, \ldots$ made of:
  - type variables $\alpha, \beta, \ldots \in \mathcal{V}$
  - type constants $\mathrm{bool}, \mathrm{int}, \mathrm{float}, \ldots \in \mathcal{B}$
  - function types $S \to T$, where S and T are type schema

Type substitutions $\theta, \rho, \sigma, \ldots$ are finite maps from $\mathcal{V}$ to $\mathcal{S}$.

Lemma: if $\Gamma \vdash v : V$ then, for all type substitution ρ, $\Gamma\rho \vdash v : V\rho$

Type schema compatibility

Two type schema S and T (with distinct type variables) are compatible if there is a type substitution ρ such that $S\rho = T\rho$.

This is unification (Jacques Herbrand, 1930) !

Unification problem

- a unification constraint is a pair of type schema (S, T)
- a unification problem is a set of unification constraints
- a solution to a unification problem $\{(S_1, T_1,) \ldots, (S_n, T_n)\}$ is a substitution ρ such that $S_1\rho = T_1\rho, \ldots, S_n\rho = T_n\rho$
- a unification problem is in solved form if it is of the form $\{(\alpha_1, T_1), \ldots, (\alpha_n, T_n)\}$ and, for all $i \leq j$, $\alpha_i \notin T_j$

Remark: a solved form <u>is</u> a substitution

Unification algorithm

A configuration is a pair of problems (C, D) with D in solved form.

Rewrite the initial configuration $(C, \emptyset)$ as much as possible by using the following rules:

$$\begin{array}{rcl}
\{(S,S)\} \cup C, D & \mapsto & C, D \\
\{(S_1 \to T_1, S_2 \to T_2)\} \cup C, D & \mapsto & \{(S_1, S_2), (T_1, T_2)\} \cup C, D \\
\{(\alpha, S)\} \cup C, D & \mapsto & C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \notin S \\
\{(S, \alpha)\} \cup C, D & \mapsto & C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \notin S \\
C, D & \mapsto & \mathrm{error} \text{ otherwise}
\end{array}$$

Example

Let $C = \{(\alpha, \beta \to \gamma), (\gamma \to \beta, \beta \to \gamma)\}$.

$$
\begin{array}{l}
\quad C, \emptyset \\
\mapsto \{(\gamma \to \beta, \beta \to \gamma)\}, \{(\alpha, \beta \to \gamma)\} \\
\mapsto \{(\gamma, \beta), (\beta, \gamma)\}, \{(\alpha, \beta \to \gamma)\} \\
\mapsto \{(\beta, \beta)\}, \{(\gamma, \beta), (\alpha, \beta \to \beta)\} \\
\mapsto \{(\gamma, \beta), (\alpha, \beta \to \beta)\}
\end{array}
$$

Unification algorithm

A configuration is a pair of problems (C, D) with D in solved form.

Rewrite the initial configuration $(C, \emptyset)$ as much as possible by using the following rules:

$$
\begin{array}{rcl}
\{(S,S)\} \cup C, D & \mapsto & C, D \\
\{(S_1 \rightarrow T_1, S_2 \rightarrow T_2)\} \cup C, D & \mapsto & \{(S_1, S_2), (T_1, T_2)\} \cup C, D \\
\{(\alpha, S)\} \cup C, D & \mapsto & C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \notin S \\
\{(S, \alpha)\} \cup C, D & \mapsto & C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \notin S \\
C, D & \mapsto & \mathrm{error} \text{ otherwise}
\end{array}
$$

Correctness: if $(C, \emptyset) \mapsto^* (\emptyset, \theta)$, then θ is a solution of C.

Completeness: if ρ is a solution of C, then there are θ and σ such that $(C, \emptyset) \mapsto^* (\emptyset, \theta)$ and $\rho = \theta\sigma$ (ρ is an instance of θ).

Application to type inference

Problem: given v, is it decidable whether there exists Γ and V such that $\Gamma \vdash v : V$?

We first define a computable function ϕ which, given Γ, v and V such that $\mathrm{FV}(v) \subseteq \mathrm{dom}(\Gamma)$, returns a unification problem:

- $\phi(\Gamma, x, V) = \{(\Gamma(x), V)\}$
- $\phi(\Gamma, \lambda x.t, V) = \{(V, \alpha \to \beta)\} \cup \phi(\Gamma \cup \{(x, \alpha)\}, t, \beta)$ (α, β fresh)
- $\phi(\Gamma, us, V) = \phi(\Gamma, u, \alpha \to V) \cup \phi(\Gamma, s, \alpha)$ (α fresh)

Correctness: if ρ satisfies $\phi(\Gamma, v, V)$, then $\Gamma\rho \vdash v : V\rho$.

Completeness: if $\Gamma\rho \vdash v : V\rho$, then ρ can be extended into a solution of $\phi(\Gamma, v, V)$.

Type inference

Problem: given v, is it decidable whether there exists Γ and V such that $\Gamma \vdash v : V$?

Assume that $\mathrm{FV}(v) = \{x_1, \ldots, x_n\}$.

- Let $\Delta = \{(x_1, \alpha_1), \ldots, (x_n, \alpha_n)\}$ with $\alpha_1, \ldots, \alpha_n \neq$ variables.
- Let β be a fresh type variable.

Then, let:

- $\psi(v) = (\Delta\theta, \beta\theta)$ if $\phi(\Delta, v, \beta)$ has most general solution θ
- $\psi(v) = \mathrm{error}$ otherwise

Correctness: if $\psi(v) = (\Delta, S)$, then $\Delta \vdash v : S$.

Completeness: if $\Gamma \vdash v : V$, then there are Δ, S and σ such that $\psi(v) = (\Delta, S)$, $\Delta\sigma \subseteq \Gamma$ and $S\sigma = V$.

Example

Is $\lambda x.xx$ typable ?

$$
\begin{aligned}
&\phi(\emptyset, \lambda x.xx, \beta)\\
=\ &\{(\beta, \alpha \to \gamma)\} \cup \phi(\{(x, \alpha)\}, xx, \gamma)\\
=\ &\{(\beta, \alpha \to \gamma)\} \cup \phi(\{(x, \alpha)\}, x, \delta \to \gamma) \cup \phi(\{(x, \alpha)\}, x, \delta)\\
=\ &\{(\beta, \alpha \to \gamma), (\alpha, \delta \to \gamma), (\alpha, \delta)\}
\end{aligned}
$$

$$
\begin{aligned}
&(\phi(\emptyset, \lambda x.xx, \beta), \emptyset)\\
\mapsto\ &(\{(\alpha, \delta \to \gamma), (\alpha, \delta)\}, \{(\beta, \alpha \to \gamma)\})\\
\mapsto\ &(\{(\delta \to \gamma, \delta)\}, \{(\alpha, \delta \to \gamma), (\beta, (\delta \to \gamma) \to \gamma)\})\\
\mapsto\ &\mathrm{error}
\end{aligned}
$$

because there is no type T such that $T = T \to S = (T \to S) \to S$
$= ((T \to S) \to S) \to S = (((T \to S) \to S) \to S) \to S = \dots$