Introduction to the simply-typed $\lambda$-calculus

Frédéric Blanqui

INRIA

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

Introduction to the simply-typed \(\lambda\)-calculus

Outline

Simply-typed \(\lambda\)-terms

Decidability of type-checking (Church approach)

Type inference (Curry approach)
Using the untyped $\lambda$-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

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

A simple type is either:

- a type constant bool, int, float, ... ∈ B
- a function type $S \to T$, where $S$ and $T$ are themselves types

Remark: every type is of the form

$$T_1 \to \ldots \to T_n \to B$$

where $n \geq 0$ and $B \in B$. 
**Order of a type**

\[
\text{order}(T_1 \to \ldots \to T_n \to B) = \max(\{0\} \cup \{1 + \text{order}(T_i) \mid 1 \leq i \leq n\})
\]

**Examples:**

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

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 $\lambda$-term

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

`bool \to bool, int \to int, (int \to int) \to (int \to int), \ldots` 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 $\lambda$-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 $\mathcal{T}$

$$\text{dom}(\Gamma) = \{x \in \mathcal{X} | \exists T, (x, T) \in \Gamma\}$$

is the domain of $\Gamma$
Assigning a type to a λ-term - Church approach

Typing $\vdash$ is inductively defined as the smallest relation on $E \times L \times T$ 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 $x \notin \text{dom}(\Gamma)$ and $(\Gamma \cup \{(x, S)\}, t, T) \in \vdash$

(app) $(\Gamma, us, T) \in \vdash$ if there is $S \in T$ such that $(\Gamma, u, S \rightarrow T) \in \vdash$ and $(\Gamma, s, S) \in \vdash$
Assigning a type to a $\lambda$-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 $E \times L \times T$ such that:

1. **(var)** \[ \Gamma \vdash x : S \quad \text{if} \quad (x, S) \in \Gamma \]

2. **(abs)** \[ \Gamma \vdash \lambda x^S.t : S \rightarrow T \quad \text{if} \quad x \notin \text{dom}(\Gamma) \]

3. **(app)** \[ \Gamma \vdash u : S \rightarrow T \quad \Gamma \vdash s : S \]

\[ \Gamma \vdash us : T \]
Example

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

\[
\begin{align*}
\Gamma \vdash \leq : \text{int} \to \text{int} \to \text{bool} & \quad \Gamma \vdash x : \text{int} \\
\Gamma \vdash x \leq : \text{int} \to \text{bool} & \quad \Gamma \vdash 2 : \text{int} \\
\Gamma \vdash x \leq 2 : \text{bool} & \quad \Gamma \vdash x \leq 2 : \text{bool} \\
\Gamma \vdash \text{if then else} : \text{bool} \to \text{int} \to \text{int} \to \text{int} & \quad \Gamma \vdash x \leq 2 : \text{bool} \\
\Gamma \vdash \text{if } x \leq 2 \text{ then else } : \text{int} \to \text{int} \to \text{int} & \\
\Gamma \vdash \text{if } x \leq 2 \text{ then } t \text{ else } u : \text{int} \\
\Gamma \vdash \lambda x. \text{if } x \leq 2 \text{ then } t \text{ else } u : \text{int} \to \text{int}
\end{align*}
\]
Assigning a type to a \( \lambda \)-term - Curry approach

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

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

\[ \Gamma \cup \{(x, S)\} \vdash t : T \]

(abs) \[ \Gamma \vdash \lambda x.t : S \rightarrow T \] if \( x \notin \text{dom}(\Gamma) \)
Outline

Simply-typed $\lambda$-terms

Decidability of type-checking (Church approach)

Type inference (Curry approach)
Decidability of type-checking

Problem: given $\Gamma$, $\nu$ and $V$, is it decidable whether $\Gamma \vdash \nu : V$?
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 $x \notin \text{dom}(\Gamma)$,
  then there is $T$ such that $V = S \rightarrow T$ and $\Gamma \cup \{(x, S)\} \vdash t : T$.

- If $v = us$,
  then there is $S$ such that $\Gamma \vdash u : S \rightarrow V$ 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)**

If $(x, S) \in \Gamma$. We have $v = x$ and $V = S$. By inversion, $(x, V') \in \Gamma$. Since $\Gamma$ is a function, $V = V'$.

**(!abs)**

If $x \notin \text{dom}(\Gamma)$. 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)**

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

We first define a computable function $\phi$ which, given $\Gamma$ and $v$, returns the unique type of $v$ in $\Gamma$ if it exists, and error otherwise.

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

Then, the following function answers the problem:

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

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

Proof. By induction on $\nu$.

- $\nu \in X$. By assumption, $(\nu, V) \in \Gamma$. Thus, $\Gamma \vdash \nu : V$.

- $\nu = \lambda x^S.t$. By assumption, $\phi(\Gamma \cup \{(x, S)\}, t) = T \neq \text{error}$ and $V = S \rightarrow T$. By IH, $\Gamma \cup \{(x, S)\} \vdash t : T$. Thus, $\Gamma \vdash \nu : V$.

- $\nu = us$. By assumption, there is $S$ such that $\phi(\Gamma, u) = S \rightarrow T$ and $\phi(\Gamma, s) = S$. By IH, $\Gamma \vdash u : S \rightarrow T$ and $\Gamma \vdash s : S$. Thus, $\Gamma \vdash \nu : V$.

Corollary: if $\psi(\Gamma, \nu, V) = \text{true}$, then $\Gamma \vdash \nu : V$. 
Completeness

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

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

**(var)**

\[
\frac{\Gamma \vdash x : S}{\phi(\Gamma, v) = V} \quad \text{if } (x, S) \in \Gamma. \text{ We have } v = x \text{ and } V = S. \text{ Thus,}
\]

**(abs)**

\[
\frac{\Gamma \vdash \lambda x^S.t : S \rightarrow T}{\Gamma \vdash \lambda x^S.t : S \rightarrow T} \quad \text{if } x \notin \text{dom}(\Gamma). \text{ We have } v = \lambda x^S.t \text{ and } V = S \rightarrow T. \text{ By IH, } \phi(\Gamma \cup \{(x, S)\}, t) = T \neq \text{error}. \text{ Thus,}
\]

**(app)**

\[
\frac{\Gamma \vdash u : S \rightarrow T \quad \Gamma \vdash s : S}{\Gamma \vdash us : T} \quad \text{. We have } v = us \text{ and } V = T. \text{ By IH,}
\]

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

Simply-typed $\lambda$-terms

Decidability of type-checking (Church approach)

Type inference (Curry approach)
Problem: given $\nu$, is it decidable whether there exists $\Gamma$ and $V$ such that $\Gamma \vdash \nu : V$?

Problem for completeness: $\nu$ has no unique type. . .

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

Idea: use type variables!

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

Types and typing are extended with type variables:

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

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

**Lemma:** if $\Gamma \vdash \nu : V$ then, for all type substitution $\rho$, $\Gamma \rho \vdash \nu : V\rho$
Type schema compatibility

Two type schema $S$ and $T$ (with distinct type variables) are compatible if there is a type substitution $\rho$ 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 \(\rho\) 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 is 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:

\[
\begin{align*}
\{(S, S)\} \cup C, D & \rightarrow C, D \\
\{(S_1 \rightarrow T_1, S_2 \rightarrow T_2)\} \cup C, D & \rightarrow \{(S_1, S_2), (T_1, T_2)\} \cup C, D \\
\{(\alpha, S)\} \cup C, D & \rightarrow C^{S}_{\alpha}, \{(\alpha, S)\} \cup D^{S}_{\alpha} \text{ if } \alpha \notin S \\
\{(S, \alpha)\} \cup C, D & \rightarrow C^{S}_{\alpha}, \{(\alpha, S)\} \cup D^{S}_{\alpha} \text{ if } \alpha \notin S \\
C, D & \rightarrow \text{error otherwise}
\end{align*}
\]
Example

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

\[
\begin{align*}
C, \emptyset & \mapsto \{ (\gamma \to \beta, \beta \to \gamma) \}, \{ (\alpha, \beta \to \gamma) \} \\
& \mapsto \{ (\gamma, \beta), (\beta, \gamma) \}, \{ (\alpha, \beta \to \gamma) \} \\
& \mapsto \{ (\beta, \beta) \}, \{ (\gamma, \beta), (\alpha, \beta \to \beta) \} \\
& \mapsto \{ (\gamma, \beta), (\alpha, \beta \to \beta) \}
\end{align*}
\]
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:

\[
\begin{align*}
\{(S, S)\} \cup C, D & \rightarrow C, D \\
\{(S_1 \rightarrow T_1, S_2 \rightarrow T_2)\} \cup C, D & \rightarrow \{(S_1, S_2), (T_1, T_2)\} \cup C, D \\
\{(\alpha, S)\} \cup C, D & \rightarrow C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \not\in S \\
\{(S, \alpha)\} \cup C, D & \rightarrow C_\alpha^S, \{(\alpha, S)\} \cup D_\alpha^S \text{ if } \alpha \not\in S \\
C, D & \rightarrow \text{error otherwise}
\end{align*}
\]

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

Completeness: if \(\rho\) is a solution of \(C\), then there are \(\theta\) and \(\sigma\) such that \((C, \emptyset) \rightarrow^* (\emptyset, \theta)\) and \(\rho = \theta\sigma\) (\(\rho\) is an instance of \(\theta\)).
Application to type inference

Problem: given ν, is it decidable whether there exists Γ and V such that Γ ⊢ ν : V ?

We first define a computable function φ which, given Γ, ν and V such that FV(ν) ⊆ dom(Γ), returns a unification problem:

\[ \phi(Γ, x, V) = \{(Γ(x), V)\} \]
\[ \phi(Γ, \lambda x.t, V) = \{(V, \alpha \rightarrow \beta)\} \cup \phi(Γ \cup \{(x, \alpha)\}, t, \beta) \quad (\alpha, \beta \text { fresh}) \]
\[ \phi(Γ, us, V) = \phi(Γ, u, \alpha \rightarrow V) \cup \phi(Γ, s, \alpha) \quad (\alpha \text { fresh}) \]

Correctness: if ρ satisfies φ(Γ, ν, V), then Γρ ⊢ ν : Vρ.

Completeness: if Γρ ⊢ ν : Vρ, then ρ can be extended into a solution of φ(Γ, ν, V).
Type inference

**Problem:** given \( v \), is it decidable whether there exists \( \Gamma \) and \( V \) such that \( \Gamma \vdash v : V \) ?

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

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

Then, let:

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

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

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

Is \( \lambda x.xx \) typable?

\[
\phi(\emptyset, \lambda x.xx, \beta)
= \{(\beta, \alpha \rightarrow \gamma)\} \cup \phi(\{(x, \alpha)\}, xx, \gamma)
= \{(\beta, \alpha \rightarrow \gamma)\} \cup \phi(\{(x, \alpha)\}, x, \delta \rightarrow \gamma) \cup \phi(\{(x, \alpha)\}, x, \delta)
= \{(\beta, \alpha \rightarrow \gamma), (\alpha, \delta \rightarrow \gamma), (\alpha, \delta)\}
\]

\[
(\phi(\emptyset, \lambda x.xx, \beta), \emptyset)
\mapsto ((\{(\alpha, \delta \rightarrow \gamma), (\alpha, \delta)\}, \{(\beta, \alpha \rightarrow \gamma)\}))
\mapsto ((\{(\delta \rightarrow \gamma, \delta)\}, \{(\alpha, \delta \rightarrow \gamma), (\beta, (\delta \rightarrow \gamma) \rightarrow \gamma)\}))
\mapsto \text{error}
\]

because there is no type \( T \) such that \( T = T \rightarrow S = (T \rightarrow S) \rightarrow S = ((T \rightarrow S) \rightarrow S) \rightarrow S = \ldots \)