Typed Lambda Calculus
Motivations

- Partial specification of programs
- Avoid meaningless programs ($1 + true$)
- Avoid memory violation
- Avoid programs with undefined semantics
Contents

- Monomorphic Types
  - Church-style
  - Curry-style
    - Type Inference
- Polymorphic Types
  - Church-style
  - Curry-style
    - Type Inference
Simply Typed Lambda Calculus
Curry-Howard Isomorphism

**Logical system** ⇔ **Language**
- Propositions ⇔ Types
- Proofs ⇔ Programs
- Proof normalisation ⇔ Program Evaluation
Curry’58, Howard’68
Adding (simply) types to $\lambda$-calculus

Grammar for types:

$$A, B ::= b \quad \text{(base types)} \quad | \quad A \rightarrow B \quad \text{(functional types)}$$

Example:

$$\text{int} \rightarrow \text{bool} \quad \text{bool} \rightarrow (\text{bool} \rightarrow \text{int}) \quad (\text{bool} \rightarrow \text{bool}) \rightarrow \text{int}$$

Remark

- $\rightarrow$ is right-associative, e.g. $A_1 \rightarrow A_2 \rightarrow A_3$ abbreviates $A_1 \rightarrow (A_2 \rightarrow A_3)$.
- Every type $A$ can be written as $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow b$, where $A_1, \ldots, A_n$ ($n \geq 0$) are arbitrary types and $b$ is a base type.
- The standard order between types is given by $A < A \rightarrow B$ and $B < A \rightarrow B$.
  Thus base types are minimal with respect this order.
- In a typed framework substitutions are always well-typed, i.e, $t\{x\backslash u\}$ means that $x$ and $u$ have the same type.
Typing Environment

- A **typing environment** \( \Gamma \) is a **finite** function from variables to types, usually written \( x_1 : A_1, \ldots, x_n : A_n \).
- Thus for example, \( x : A, y : B \) and \( y : B, x : A \) are two different notations for the same typing environment.
- The **domain** of \( \Gamma = x_1 : A_1, \ldots, x_n : A_n \), written \( \text{dom}(\Gamma) \), is the set \( \{x_1, \ldots, x_n\} \).
- We write \( \Gamma, x : A \) for the typing environment extending \( \Gamma \) with the pair \( x : A \). It is only defined iff \( x \notin \text{dom}(\Gamma) \).
Typed Lambda Calculus in Church-Style

\[
\begin{array}{c}
\Gamma, x : A \vdash x : A \\
\hline
(ax)
\end{array}
\]

\[
\begin{array}{c}
\Gamma, x : A \vdash t : B \\
\hline
\Gamma \vdash \lambda x : A.t : A \rightarrow B \\
(\rightarrow i)
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash t : A \rightarrow B \\
\hline
\Gamma \vdash u : A \\
\hline
\Gamma \vdash t \ u : B \\
(\rightarrow e)
\end{array}
\]

We can also add constants: each constant \(c\) has an associated type \(TC(c) : A\).

\[
\begin{array}{c}
\Gamma \vdash c : TC(c) \\
\end{array}
\]

We can also add the let constructor:

\[
\begin{array}{c}
\Gamma \vdash t : A \\
\hline
\Gamma, x : A \vdash u : B \\
\hline
\Gamma \vdash \text{let } x : A = t \text{ in } u : B
\end{array}
\]

We denote by \(\Gamma \vdash_C t : A\) the derivability/typing relation. We say that \(t\) is typable in Church-Style iff there is \(\Gamma\) and \(A\) such that \(\Gamma \vdash_C t : A\).
Examples

Example of typable term in an empty environment:

\[
\begin{align*}
& y : A \to A \vdash y : A \to A \\
& \vdash \lambda y : A \to A. y : (A \to A) \to (A \to A) \\
& x : A \vdash x : A \\
& \vdash \lambda x : A. x : A \to A \\
& \vdash (\lambda y : A \to A. y)(\lambda x : A. x) : A \to A
\end{align*}
\]

Example of typable term in a non-empty environment:

\[
\begin{align*}
& c_1 : int \to int \vdash c_1 : int \to int \\
& c_1 : int \to int \vdash c_1 : int \to int \\
& c_1 : int \to int \vdash c_1 : int \to int \vdash 3 : int \\
& c_1 : int \to int \vdash c_1 : int \to int \\
& c_1 int \to int \vdash c_1 3 : int \\
& c_1 : int \to int \vdash c_1 (c_1 3) : int
\end{align*}
\]

Example of non-typable term: \( \lambda x.xx \).
Typed Properties

[Unicity]: If $\Gamma \vdash_C t : A$ and $\Gamma \vdash_C t : B$, then $A \equiv B$.

Proof.
By induction on $t$. □

[Weakening and Strengthening]: Let $\Gamma = \{x : B \mid x \in \text{fv}(t)\}$ and $\Gamma \subseteq \Delta_1 \subseteq \Delta_2$. Then $\Delta_1 \vdash_C t : A$ iff $\Delta_2 \vdash_C t : A$.

Proof.
By induction on $t$. □

[Subject Reduction] If $\Gamma \vdash_C t : A$ and $t \rightarrow_\beta t'$, then $\Gamma \vdash_C t' : A$.

Proof.
By induction on $\Gamma \vdash_C t : A$ (blackboard). □
Typing Algorithm

\[
\begin{align*}
\text{Type(}\Gamma, c) & = \text{TC}(c) \\
\text{Type(}\Gamma, x) & = A \quad \text{if } x : A \in \Gamma \\
\text{Type(}\Gamma, \lambda x: A. t) & = A \rightarrow B \quad \text{if } \text{Type(}(\Gamma, x : A), t) = B \\
\text{Type(}\Gamma, t \ u) & = B \quad \text{if } \text{Type(}\Gamma, t) = A \rightarrow B \text{ and } \text{Type(}\Gamma, u) = A \\
\text{Type(}\Gamma, \text{let } x: A = t \text{ in } u) & = B \quad \text{if } \text{Type(}\Gamma, t) = A \text{ and } \text{Type(}(\Gamma, x : A), u) = B \\
\text{Type(}\Gamma, t) & = \text{error} \quad \text{otherwise}
\end{align*}
\]
Properties of the Typing Algorithm

[Termination]
For every term $t$ and every environment $\Gamma$, the call $\text{Type}(\Gamma, t)$ terminates.

[Soundness]
If $\text{Type}(\Gamma, t) = A$, then $\Gamma \vdash_C t : A$.

[Completeness]
If $\Gamma \vdash_C t : A$, then $\text{Type}(\Gamma, t) = A$.

Said differently:
If $\text{Type}(\Gamma, t) = \text{erreur}$, then $t$ is not typable in $\Gamma$. 
Typed Lambda Calculus in Curry-Style

\[
\frac{\Gamma, x : A \vdash x : A}{(ax)}
\]

\[
\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \quad (\rightarrow i)
\]

\[
\frac{\Gamma \vdash t : A \rightarrow B}{\Gamma \vdash \lambda x.t : A \rightarrow B}
\]

\[
\frac{\Gamma \vdash u : A}{\Gamma \vdash \lambda u.t : A \rightarrow B} \quad (\rightarrow e)
\]

We can also add constants: each constant \( c \) has an associated type \( TC(c) : A \).

\[
\frac{\Gamma \vdash c : TC(c)}{}
\]

We can also add lets:

\[
\frac{\Gamma \vdash t : A \quad \Gamma, x : A \vdash u : B}{\Gamma \vdash \text{let } x = t \text{ in } u : B}
\]

We denote by \( \Gamma \vdash_{\lambda} t : A \) the derivability/typing relation. We say that \( t \) is typable in Curry-Style iff there is \( \Gamma \) and \( A \) such that \( \Gamma \vdash_{\lambda} t : A \).
Properties

Unicity does not hold anymore:

\[ \Gamma \vdash \lambda x. x : \text{int} \rightarrow \text{int} \quad \Gamma \vdash \lambda x. x : \text{bool} \rightarrow \text{bool} \]

The identity function behaves in the same way for \text{int} and \text{bool}:

Polymorphism
Difficulties for a Typing Algorithm

\[ \text{Type}(\Gamma, \lambda x.t) = A \rightarrow B \quad \text{if there exists } A \text{ s.t. } \text{Type}((\Gamma, x : A), T) = B \]

\[ \text{Type}(\Gamma, \text{let } x = t \text{ in } u) = B \quad \text{if there exists } A \text{ s.t. } \text{Type}(\Gamma, t) = A \text{ and } \text{Type}((\Gamma, x : A), u) = B \]
The type inference problem for a term $t$: $\exists \Gamma \exists A$ such that $\Gamma \vdash_{\lambda} t : A$?
Towards a Monomorphic Type Inference Algorithm

General Technical Tools:

- We consider a table of types for constants, called $TC$. E.g. $TC(3) = \text{int}$ and $TC(c_1) = \text{int} \rightarrow \text{int}$.
- Let $t$ be a term. For each sub-term $u$ of $t$ we introduce a type variable $\alpha_u$.
- We associate to every term $t$ a set of equations $SE(t) = \{\alpha_1 \doteq u_1, \ldots, \alpha_m \doteq u_m\}$.
- The solution to the type inference problem for the term $t$ will be given by the most general unifier (mgu) of the corresponding set of equations.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$SE(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>${\alpha_t \doteq \alpha_x}$</td>
</tr>
<tr>
<td>$c$</td>
<td>${\alpha_t \doteq TC(c)}$</td>
</tr>
<tr>
<td>$uv$</td>
<td>${\alpha_u \doteq \alpha_v \rightarrow \alpha_t} \cup SE(u) \cup SE(v)$</td>
</tr>
<tr>
<td>$\lambda x. u$</td>
<td>${\alpha_t \doteq \alpha_x \rightarrow \alpha_u} \cup SE(u)$</td>
</tr>
<tr>
<td>let $x = u$ in $v$</td>
<td>${\alpha_t \doteq \alpha_v; \alpha_x \doteq \alpha_u} \cup SE(u) \cup SE(v)$</td>
</tr>
</tbody>
</table>
Examples

- \( t_0 := \lambda f. \lambda g. f g \)
  \[ SE(t_0) = \{ \alpha \vdash \alpha f \to \alpha \lambda g.f g, \alpha \lambda g.f g \vdash \alpha g \to \alpha f g, \alpha f \vdash \alpha g \to \alpha f g, \alpha f \vdash \alpha f, \alpha g \vdash \alpha g \} \]
  - The mgu of \( SE(t_0) \) is the substitution \( \sigma_{t_0} \) such that \( \sigma_{t_0}(\alpha_{t_0}) = (\alpha g \to \alpha f g) \to (\alpha g \to \alpha f g) \).
  - \( t_0 \) is typable with instances of \( \sigma_{t_0}(\alpha_{t_0}) \) which are types of the form \((A \to B) \to (A \to B)\), for any arbitrary types \(A\) and \(B\).

- \( t_1 := \text{let } f = \lambda x. x \text{ in } f(f z) \)
  \[ SE(t_1) = \{ \alpha \vdash \alpha f(f z), \alpha f \vdash \alpha \lambda x.x, \alpha \lambda x.x \vdash \alpha x \to \alpha x, \alpha x \vdash \alpha x, \alpha f \vdash \alpha f z \to \alpha f(f z), \alpha f \vdash \alpha f, \alpha f \vdash \alpha z \to \alpha f z, \alpha z \vdash \alpha z \} \]
  - The mgu of \( SE(t_1) \) is the substitution \( \sigma_{t_1} \) such that \( \sigma_{t_1}(\alpha_{t_1}) = \alpha f(f z) \).
  - \( t_1 \) is typable with instances of \( \sigma_{t_1}(\alpha_{t_1}) \) which are arbitrary types \(A\).
Properties of the Type Inference Algorithm

**Theorem (Soundness)** If $\sigma$ is a solution of $S E(t)$, then $t$ is (Curry) typable. Moreover, $\Delta \vdash_\lambda t : \sigma'(\alpha_t)$, where $\Delta = \{ x : \sigma'(\alpha_x) \mid x \in \text{fv}(t) \}$ and $\sigma'$ is an instance of $\sigma$.

**Theorem (Completeness)** If $t$ is (Curry) typable, then $S E(t)$ admits a solution.

**Theorem (Principality)** If $t$ is (Curry) typable, i.e. $\Delta \vdash_\lambda t : A$, then $A$ is an instance of the principal type (i.e. $A = \sigma'(\sigma(\alpha_t))$), where $\sigma$ is the mgu of system $S E(t)$ and $\sigma'$ is a substitution.
Polymorphic Lambda Calculus
Motivations

- General behaviour of an operator, i.e. without looking at the particular nature (or type) of its parameters.
- More clear and concise programs.
- Reutilisation of operators.
Different Forms of Polymorphism

- Ad-hoc polymorphism (overloaded constructors), originally described by Strachey.
- Subtyping polymorphism, introduced by Wegner and Cardelli: $\Gamma \vdash t : A \rightarrow B$, $\Gamma \vdash u : A'$ et $A' \leq A$ alors $tu : B$.
- Parametric polymorphism, introduced by Reynolds and Girard.
Parametric Polymorphism

Motivation:
Many identity functions:
\[ Id_{\text{int}} : \text{int} \to \text{int}, \quad Id_{\text{bool}} : \text{bool} \to \text{bool}, \quad Id_{\text{int} \to \text{int}} : (\text{int} \to \text{int}) \to (\text{int} \to \text{int}), \ldots \]
Many functions to add an element to a list:
\[ A_{\text{int}} : \text{int} \to \text{list}(\text{int}) \to \text{list}(\text{int}), \quad A_{\text{bool}} : \text{bool} \to \text{list}(\text{bool}) \to \text{list}(\text{bool}), \ldots \]
Idea:
\[ Id : \forall \alpha. \alpha \to \alpha \]
and thus:
\[ Id_{\text{int}} = Id[\text{int}] \]
\[ Id_{\text{bool}} = Id[\text{bool}] \]
\[ Id_{\text{int} \to \text{int}} = Id[\text{int} \to \text{int}] \]
Remark The key idea is the instantiation relation between the general type \( \alpha \), and the particular used types \text{int}, \text{bool}, \text{int} \to \text{int}. \]
Girard-Reynolds Polymorphism

**Types**: \( A ::= b \mid \alpha \mid A \to A \mid \forall \alpha. A \)

**Notation**: \( \forall \alpha.\forall \beta. A = \forall (\alpha, \beta). A \).

**Expressions**:

\[
\begin{align*}
t ::= & \ x \mid c \mid t \ t \mid \\
& \lambda x : A. t \mid \text{let } x : A = t \text{ in } t \mid \\
& t[A] \mid \Lambda \alpha t
\end{align*}
\]

**Reduction Rules**:

\[
\begin{align*}
(\lambda x : A. t) u & \rightarrow t\{x\backslash u\} \\
\text{let } x : A = u \text{ in } t & \rightarrow t\{x\backslash u\} \\
(\Lambda \alpha t)[A] & \rightarrow t\{\alpha\backslash A\}
\end{align*}
\]

**Reduction Rules**:

Let \( Id = \Lambda \alpha \lambda x : \alpha. x \)

\[
\begin{align*}
Id[\text{int}] & \rightarrow \lambda x : \text{int}. x \\
(Id[\text{int}]) 3 & \rightarrow (\lambda x : \text{int}. x) 3 \rightarrow 3
\end{align*}
\]
Type Free Variables

By induction on types:

\[
\begin{align*}
\text{tfv}(b) &= \emptyset \\
\text{tfv}(\alpha) &= \{\alpha\} \\
\text{tfv}(\forall\alpha.A) &= \text{tfv}(A) \setminus \{\alpha\} \\
\text{tfv}(A \rightarrow B) &= \text{tfv}(A) \cup \text{tfv}(B)
\end{align*}
\]

A type \( A \) is closed iff \( \text{tfv}(A) = \emptyset \).

Example: \( \text{tfv}(\beta \rightarrow (\forall\alpha.\alpha \rightarrow \gamma)) = \{\beta, \gamma\} \) and \( \forall\alpha.\forall\beta.(\alpha \rightarrow \beta) \) is closed.
Polymorphic Lambda Calculus in Church-Style (Girard)

The same rules used for monomorphic Church-style plus:

\[
\frac{\Gamma \vdash t : A \quad \alpha \notin \text{tfv}(\Gamma)}{\Gamma \vdash \Lambda \alpha t : \forall \alpha . A}
\]

\[
\frac{\Gamma \vdash t : \forall \alpha . A}{\Gamma \vdash t[B] : A{\alpha \backslash B}}
\]

where \( \text{tfv}(\Gamma) := \bigcup_{x \in \Gamma} \text{tfv}(\Gamma(x)) \) is the set of type free variables of \( \Gamma \).

We denote by \( \Gamma \vdash_G t : A \) the derivability/typing relation.

Example:

\[
\frac{x : \alpha \vdash x : \alpha}{\vdash \lambda x : \alpha . x : \alpha \to \alpha}
\]

\[
\vdash \Lambda \alpha \lambda x : \alpha . x : \forall \alpha . (\alpha \to \alpha)
\]

\[
\vdash (\Lambda \alpha \lambda x : \alpha . x)[\text{int}] : \text{int} \to \text{int} \quad \vdash 3 : \text{int}
\]

\[
\vdash ((\Lambda \alpha \lambda x : \alpha . x)[\text{int}]) 3 : \text{int}
\]
Theorem

(Schubert) The inference problem in Girard system is \textit{undecidable}.

\textbf{Idea to become decidable:} to restrict the grammar of types.
**Type matrix** : \( M ::= b \mid \alpha \mid M \rightarrow M \)

**ML Type** : \( S ::= \forall \alpha_1 \ldots \forall \alpha_n. M \)

A type matrix is a particular case of type.

**Type for constants** :

\[
\begin{align*}
TC(+) & : \quad \text{int} \rightarrow \text{int} \rightarrow \text{int} \\
TC(\text{fst}) & : \quad \forall \alpha \forall \beta. (\alpha \rightarrow \beta) \rightarrow \alpha \\
TC(\text{snd}) & : \quad \forall \alpha \forall \beta. (\alpha \rightarrow \beta) \rightarrow \beta \\
TC(\text{ifthenelse}) & : \quad \forall \alpha. (\text{bool} \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \\
TC(\text{fix}) & : \quad \forall \alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha
\end{align*}
\]
Definition

An ML type \( A \) is an instance of an ML type \( \forall \alpha_1 \ldots \forall \alpha_n. B \), written \( A \leq \forall \alpha_1 \ldots \forall \alpha_n. B \) iff there exist \( C_1, \ldots, C_n \) s.t. \( A = B[\alpha_1, \ldots, \alpha_n \setminus C_1, \ldots, C_n] \). In particular, for \( n = 0 \), we have \( A \leq B \) iff \( A \equiv B \).

Example:

\[
\begin{align*}
int \to int & \leq \forall \alpha. \alpha \to \alpha \\
bool \to int & \leq \forall \alpha. \forall \beta. \alpha \to \beta \\
bool \to int & \not\leq \forall \alpha. \alpha \to \alpha \\
\forall \beta. \text{int} \to \beta & \leq \forall \alpha. \forall \beta. \alpha \to \beta \\
\forall \beta. (\beta \to \beta) \to \forall \beta. (\beta \to \beta) & \not\leq \forall \alpha. \alpha \to \alpha
\end{align*}
\]
Type Generalization

Definition

The Gen operator is given by $\text{Gen}(A, \Gamma) = \forall \alpha_1 \ldots \forall \alpha_n. A$, where each variable $\alpha_i$ is free in $A$ but not in $\Gamma$, i.e. forall $i = 1 \ldots n$, we have $\alpha_i \in \text{tfv}(A) \setminus \text{tfv}(\Gamma)$.

Example: Let $A = \alpha \rightarrow \beta$ and $\Gamma = x : \beta, y : \forall \alpha. \alpha$. Then $\text{Gen}(A, \Gamma) = \forall \alpha. \alpha \rightarrow \beta$. 
We denote by $\Gamma \vdash_{ML} t : A$ the derivability/typing relation.
Example

Let $A = \forall \alpha. \alpha \to \alpha$

\[
\begin{align*}
  & y : \alpha \vdash y : \alpha \\
  \implies & \vdash \lambda y. y : \alpha \to \alpha \\
\end{align*}
\]

\[
\begin{align*}
  & \text{int} \to \text{int} \leq A \\
  \implies & f : A \vdash f : \text{int} \to \text{int} \\
  \implies & \vdash f : \forall \alpha. \alpha \to \alpha \\
\end{align*}
\]

\[
\begin{align*}
  & f : A \vdash 1 : \text{int} \\
  \implies & \vdash \text{let } f = \lambda y. y \text{ in } f \ 1 : \text{int} \\
\end{align*}
\]
Properties

[Substitution] :
If $\Gamma, x : \forall \alpha_1 . . . \forall \alpha_n. B \vdash_{ML} t : A$ and $\Gamma \vdash_{ML} u : B$, then $\Gamma \vdash_{ML} t\{x\{u\} : A$.

[Subject Reduction] :
If $\Gamma \vdash_{ML} t : A$ and $t \rightarrow t'$, then $\Gamma \vdash_{ML} t' : A$. 
Let $t := \text{let } x = \lambda y. y \text{ in } x$. Let consider the set of equations:

$$\{ \alpha_t \doteq \alpha_x, \alpha_x \doteq \text{Gen}(\alpha_{\lambda y.y}, \emptyset), \alpha_{\lambda y.y} \doteq \alpha_y \rightarrow \alpha_y \}$$

If we treat the second equation we obtain $\alpha_x \equiv \forall \alpha_{\lambda y.y}. \alpha_{\lambda y.y}$, which is incorrect since $\alpha_x$ should be a functional type (an arrow).
Towards a Type Inference Algorithm: Notations

Let $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ be an environment and let $\sigma$ be a type substitution. then $\sigma(\Gamma) = x_1 : \sigma(A_1), \ldots, x_n : \sigma(A_n)$.

We note $\text{inst}(\forall \alpha_1 \ldots \forall \alpha_n. A)$ the type $A\{\alpha_1, \ldots, \alpha_n \backslash \beta_1, \ldots, \beta_n\}$, where $\beta_1, \ldots, \beta_n$ are fresh variables.
Damas-Milner-Tofte Algorithm

**Input**: an environment $\Gamma$ and a term $t$ s.t. $fv(t) \subseteq \Gamma$.

**Output**: an ML type $A$ and a type substitution $\sigma$ s.t. $\sigma(\Gamma) \vdash_{ML} t : A$ ($id$ is the empty type substitution).

\[
W((\Delta, x : A), x) = (inst(A), id)
\]
\[
W(\Delta, c) = (inst(TC(c)), id)
\]
\[
W(\Delta, \lambda x. u) = (\rho_B(\alpha_x) \to B, \rho_B)
\]

where $W((\Delta, x : \alpha_x), u) = (B, \rho_B)$ and $\alpha_x$ is a fresh variable

\[
W(\Delta, u \, v) = (\mu(\alpha), \mu \circ \rho_C \circ \rho_B)
\]

where $W(\Delta, u) = (B, \rho_B)$, $W(\rho_B(\Delta), v) = (C, \rho_C)$,

$\alpha$ is a fresh variable and $\mu = mgu\{\rho_C(B) \doteq C \to \alpha\}$

\[
W(\Delta, \text{let } x = u \text{ in } v) = (C, \rho_C \circ \rho_B)
\]

where $W(\Delta, u) = (B, \rho_B)$ and $W((\rho_B(\Delta), x : \text{Gen}(B, \rho_B(\Delta))), v) = (C, \rho_C)$
Premier exemple

\[ W(\emptyset, \lambda x. \ast 4x) = (\text{int} \to \text{int}, \{\alpha/\text{int} \to \text{int}, \beta/\text{int}, \alpha_x/\text{int}\}) \]
\[ W(x : \alpha_x, \ast 4x) = (\text{int}, \{\alpha/\text{int} \to \text{int}, \beta/\text{int}, \alpha_x/\text{int}\}), \text{computes } \text{mgu}\{\text{int} \to \text{int} \simeq \alpha_x \to \beta}\] 
\[ W(x : \alpha_x, \ast 4) = (\text{int} \to \text{int}, \{\alpha/\text{int} \to \text{int}\}), \text{computes } \text{mgu}\{\text{int} \to \text{int} \to \text{int} \simeq \text{int} \to \alpha\} \]
\[ W(x : \alpha_x, \ast) = (\text{int} \to \text{int} \to \text{int}, \text{id}) \]
\[ W(x : \alpha_x, 4) = (\text{int}, \text{id}) \]
\[ W(x : \alpha_x, x) = (\alpha_x, \text{id}) \]

\[ \text{Then } \emptyset \vdash_{\text{ML}} \lambda x. \ast 4x : \text{int} \to \text{int}. \]
Second Example

\[
\begin{align*}
W(\emptyset, \text{let } f = \lambda x. x \text{ in } f^2) &= (\text{int}, \{\beta/\text{int}, \gamma/\text{int}\}) \\
W(\emptyset, \lambda x. x) &= (\alpha_x \rightarrow \alpha_x, \text{id}) \\
W(x : \alpha_x, x) &= (\alpha_x, \text{id}) \\
W(f : \forall \alpha_x. \alpha_x \rightarrow \alpha_x, f^2) &= (\text{int}, \{\beta/\text{int}, \gamma/\text{int}\}), \text{ where } \text{mgu}\{\beta \rightarrow \beta \vdash \text{int} \rightarrow \gamma\} = \{\beta/\text{int}, \gamma/\text{int}\} \\
W(f : \forall \alpha_x. \alpha_x \rightarrow \alpha_x, f) &= (\beta \rightarrow \beta, \text{id}) \\
W(f : \forall \alpha_x. \alpha_x \rightarrow \alpha_x, 2) &= (\text{int}, \text{id}) \\
\end{align*}
\]

Then \(\emptyset \vdash_{\text{ML}} \text{let } f = \lambda x. x \text{ in } f^2 : \text{int}\).
Properties of the Type Inference Algorithm

Theorem (Soundness) If \( W(\Delta, t) = (A, \sigma) \), then \( \sigma(\Delta) \vdash_{ML} t : A \).

Theorem (Completeness) Let \( \Delta = x_1 : \alpha_1, \ldots, x_n : \alpha_n \) where \( \text{fv}(t) \subseteq \{x_1, \ldots, x_n\} \). Let \( \tau \) be a type substitution, If \( \tau(\Delta) \vdash_{ML} t : B \), then \( W(\Delta, t) = (A, \sigma) \), where \( B \) is an instance of \( A \) and \( \tau \) is an instance of \( \sigma \).

Corollary: \( \emptyset \vdash_{ML} t : A \) iff \( W(\emptyset, t) = (B, \sigma) \) and \( A \) is an instance of \( B \).
Strong Normalization of Simply Typed Lambda Calculus
Typed Properties

[Strong Normalization] Every simply typed term is normalising:
if $\Gamma \vdash_\lambda t : A$, then $t \in SN_\beta$. 
Defining Strongly Normalizing Terms

- \( t \in SN_\beta \)
  - iff there is no infinite \( \beta \)-reduction sequence starting at \( t \).
- \( t \in SN_\beta \)
  - iff every \( \beta \)-reduction sequence starting at \( t \) is finite.

**First** inductive alternative:
- If \( t \) is a \( \beta \)-normal form, then \( t \in SN \)
- If \( \forall t' \ [(t \rightarrow_\beta t') \implies t' \in SN], \) then \( t \in SN \)
  (the first line is a special case of the second one)

**Second** inductive alternative:
- \( t_1, \ldots, t_n \in SN \) implies \( x \vec{t} = x t_1 \ldots t_n \in SN \).
- \( t \in SN \) implies \( \lambda x.t \in SN \).
- \( t[x\backslash u]\vec{r} \in SN \) and \( u \in SN \) implies \( (\lambda x.t)u\vec{r} \in SN \).

In both cases one shows that \( t \in SN \iff t \in SN_\beta \).

**Definition (Measuring \( SN_\beta \)-terms)**

Given \( t \in SN_\beta \), we define the **measure** \( \mu_\beta(t) \) as \( \max\{n \in \mathbb{N} \mid t \rightarrow^n_\beta t' \} \).

Note that \( t \rightarrow_\beta t' \) implies \( \mu_\beta(t') < \mu_\beta(t) \), so that \( t \in SN_\beta \) and \( t \rightarrow_\beta t' \) implies \( t' \in SN_\beta \).
Some General Remarks About $SN_{\beta}$-Terms

- $u \in SN_{\beta}$ iff $\lambda y.\ u \in SN_{\beta}$.
- $u_1, \ldots, u_n \in SN_{\beta}$ iff $x\ u_1 \ldots u_n \in SN_{\beta}$.
- In general, if $t \in SN_{\beta}$, then every subterm of $t$ is also $SN_{\beta}$, but the converse is not true, e.g. $(\lambda x.xx)(\lambda x.xx)$.
- This is because $SN_{\beta}$ is not stable by substitution. Example: $x\ x \in SN_{\beta}$, $\lambda y.y\ y \in SN_{\beta}$, but $(x\ x)\{x\ \lambda y.y\ y\} = \Delta\ \Delta \notin SN_{\beta}$. 
First Proof of the SN property

- This first proof is due to Tait.
- Uses the **first** alternative definition of $SN_\beta$
- It is based on a predicate $SC$ to characterize *strong computable* terms.

**Definition**

Let $t$ be of type $A = A_1 \rightarrow \ldots \rightarrow A_n \rightarrow \tau$. Then $t \in SC$ iff forall $u_i \in SC$ of type $A_i$ we have $t \vec{u} = t \ u_1 \ldots u_n \in SN_\beta$.

The previous definition implies

1. $SC \subseteq SN_\beta$.
2. $SC$ is closed under $\beta$ (i.e. $t \in SC$ and $t \rightarrow_\beta t'$ implies $t' \in SC$).
3. $x \in SC$ for every variable $x$ (using 1).
Lemma

If $t, u_1, \ldots, u_n \ (n \geq 1) \in SN_\beta$ and $t\{x\backslash u_1\}u_2 \ldots u_n \in SN_\beta$, then $(\lambda x.t)u_1u_2 \ldots u_n \in SN_\beta$.

Proof.

By the first alternative definition of $SN_\beta$, it is sufficient to show that all the reducts of $(\lambda x.t)u_1 \ldots u_n$ are in $SN_\beta$. We reason by induction on $\mu(t) + \Sigma_i \mu(u_i)$. Case analysis on the reducts:

- $(\lambda x.t')u_1 \ldots u_n$, where $t \rightarrow t'$. Then $\mu(t') < \mu(t)$, we conclude by the i.h.
- $(\lambda x.t)u_1 \ldots u'_i \ldots u_n$, where $u_i \rightarrow u'_i$. Then $\mu(u'_i) < \mu(u_i)$, we conclude by the i.h.
- $t\{x\backslash u_1\}u_2 \ldots u_n$. We conclude by the hypothesis.
Lemma

Let $t$ be a typed term. Let $\sigma$ be a type preserving substitution mapping all the free variables of $t$ to terms in $\mathit{SC}$. Then $t\sigma \in \mathit{SC}$.

Proof.

We proceed by induction on the typed term $t$.

- If $t = x$, then $x\sigma = \sigma(x) \in \mathit{SC}$ by hypothesis.
- If $t = uv$, then let $\vec{r} \in \mathit{SC}$. We have $u\sigma$ and $v\sigma$ in $\mathit{SC}$ by i.h. Then $(uv)\sigma\vec{r} = u\sigma \circ \sigma \vec{r} \in \mathit{SN}_\beta$ by definition.
- If $t = \lambda x.u$, then $(\lambda x.u)\sigma =_\alpha \lambda x.u\sigma$. Since $\sigma \cup \{x\} \{x\}$ verifies the hypothesis of the lemma, then by the i.h. $u(\sigma \cup \{x\}) = u\sigma \in \mathit{SC} \subseteq \mathit{SN}_\beta$. To show $\lambda x.u\sigma \in \mathit{SC}$ we show $(\lambda x.u\sigma)\vec{r} \in \mathit{SN}_\beta$ for $\vec{r} \in \mathit{SC} \subseteq \mathit{SN}_\beta$. This follows from the previous lemma.

$\square$
Lemma

Every typed term is in $SC$.

Proof.

Using the previous lemma with the identity substitution (which verifies the hypothesis of the current lemma).
Theorem

*Every typed term is in $S N_{\beta}$.*

Proof.

Using the previous lemma and the fact the $S C \subseteq S N_{\beta}$. □
Second proof of the SN property

- Can be found in Femke van Raamsdonk's Thesis.
- Uses the **second** alternative definition of $SN_\beta$

1. Define $\Lambda_A$ (terms of type $A$) inductively:
   - If $x$ is a variable of type $A$, then $x \in \Lambda_A$.
   - If $t \in \Lambda_C$ and $x$ is a variable of type $B$, then $\lambda x.t \in \Lambda_{B \to C}$.
   - If $t \in \Lambda_{B \to A}$ and $u \in \Lambda_B$, then $tu \in \Lambda_A$.

2. Define $S N_A = SN \cap \Lambda_A$.

3. Define $X \to Y = \{t | \forall u. (u \in X \text{ implies } tu \in Y)\}$.

4. Show $\Lambda_{A \to B} = \Lambda_A \to \Lambda_B$.

5. Show $S N_A \to S N_B \subseteq S N_{A \to B}$ (easy).

6. If $u \in S N_{A_1} \to S N_{A_2} \to \ldots \to S N_{A_m}$ with $A_m$ a base type and $t \in S N_B$, then $t\{x\{u\} \in S N_B$ (induction on SN using 5).

7. Show $S N_{A \to B} \subseteq S N_A \to S N_B$ (using 6).

8. Show that $\Lambda_A \subseteq S N_A$ (by induction using 7).

9. Since $S N_A \subseteq SN = S N_\beta$ we conclude.
Lemma

If $t$ and $u$ are typed and belong to $SN_\beta$, then $t\{x\backslash u\} \in SN_\beta$.

Proof.

By induction on $\langle \text{type}(u), \mu_\beta(t), \text{size}(t) \rangle$.

- The base case $\langle \text{base type}, 0, 1 \rangle$ is trivial.
- Case $t = \lambda y.v$ is by the i.h. on $v$ ($\text{size}(\_)$ strictly decreases).
- Case $t = y \vec{c}$ with $x \neq y$ is by the i.h. on $c_i$ ($\mu_\beta(\_)$ decreases and $\text{size}(\_)$ strictly decreases).
- Case $t = x$. We have $x\{x\backslash u\} = u \in SN_\beta$ by hypothesis.
- Case $t = x b \vec{c}$. By the i.h. $B = b\{x\backslash u\}$ and $C_i = c_i\{x\backslash u\}$ are in $SN_\beta$. We want to show that $u B \vec{C} \in SN_\beta$. It is sufficient (first alternative definition of $SN_\beta$) to show that all its reducts are in $SN_\beta$. We reason by induction on $\mu_\beta(u) + \mu_\beta(B) + \Sigma_i \mu_\beta(C_i)$. The reducts are
  - $u' B \vec{C}$, where $u \rightarrow u'$. Apply the i.h.
  - $u B' \vec{C}$, where $B \rightarrow B'$. Apply the i.h.
  - $u' B C_1 \ldots C_i' \ldots C_n$, where $C_i \rightarrow C'_i$. Apply the i.h.
  - $v\{y\backslash B\} \vec{C}$, where $u = \lambda y.v$. But $v\{y\backslash B\} \vec{C} = (z\vec{C})\{z\backslash v\{y\backslash B\}\}$ and $\text{type}(v\{y\backslash B\}) < \text{type}(u)$. We thus conclude by the i.h. since $z\vec{C}$ and $v\{y\backslash B\}$ are typed and in $SN_\beta$ by the i.h.

$\Box$
Case $t = (\lambda z.b) c \vec{d}$. By the i.h. $B = b{x\backslash u}$ and $C = c{x\backslash u}$ and $D_i = d_i{x\backslash u}$ are in $SN_\beta$. Suppose $t{x\backslash u} = (\lambda z.B) C \vec{D} \notin SN_\beta$. Then $B{z\backslash C} \vec{D} \notin SN_\beta$. But $B{z\backslash C} \vec{D} = (b{z\backslash c}\vec{d}){x\backslash u}$ and $\mu_\beta(b{z\backslash c}\vec{d}) < \mu_\beta(t)$. Thus $B{z\backslash C} \vec{D} \in SN_\beta$ by the i.h. Contradiction. Thus $t{x\backslash u} = (\lambda z.B) C \vec{D} \in SN_\beta$. 
Theorem
If $t$ is typable, then $t \in SN_\beta$.

Proof.
By induction on the typing derivation of $t$.

- Case $t = x$ is trivial.
- Case $t = \lambda y. u$ holds by the i.h.
- For the case $t = u \, v$ use the fact that $t = (z \, v)\{z\, u\}$ and apply previous lemma (verification of the hypothesis is easy).
Fourth proof of the SN property

See for example Gandy’s proof by Alexandre Miquel.
A combinatorial proof of strong normalisation for the simply typed lambda-calculus.
Strong Normalization of System F
Reducibility Candidates

- **Neutral Terms**: Terms that are not abstractions.
- **Strongly Normalizing Terms**: $SN_F$

**Definition**

A reducibility candidate of type $A$ is a set $R$ of terms of type $A$ such that

(CR1) If $t \in R$, then $t \in SN_F$

(CR2) If $t \in R$ and $t \rightarrow_F t'$, then $t' \in R$

(CR3) If $t$ is neutral and ($t \rightarrow_F t'$ implies $t' \in R$), then $t \in R$.

If $R$ and $S$ are reducibility candidates of type $A$ and $B$ respectively, then $R \rightarrow S$ is a set of terms of type $A \rightarrow B$ defined by

$$t \in R \rightarrow S \text{ iff } \forall u.(u \in R \text{ implies } tu \in S)$$
Remarks

- A consequence of $(CR3)$: If $t$ is neutral and normal, then $t \in \mathcal{R}$.
- $\mathcal{R}$ of type $A$ is never empty, it contains at least the variables of type $A$.
- The set $\{t \in S N_F \text{ and } t \text{ of type } A\}$ is a reducibility candidate.
- $\mathcal{R} \rightarrow S$ is a reducibility candidate.
Let $T$ be a type where $\text{tfv}(T) \subseteq \vec{\alpha}$. We write $T\{\vec{\alpha} \backslash \vec{A}\}$ for the simultaneous substitution of $\vec{\alpha}$ by $\vec{A}$. Given $\vec{R}$ a sequence of reducibility candidates, we define a set $\text{RED}_T(\vec{\alpha}, \vec{R})$ of terms of type $T\{\vec{\alpha} \backslash \vec{A}\}$.

- If $T = \alpha_i$, then $\text{RED}_T(\vec{\alpha}, \vec{R}) = \mathcal{R}_i$
- If $T = A \rightarrow B$, then $\text{RED}_{A \rightarrow B}(\vec{\alpha}, \vec{R}) = \text{RED}_A(\vec{\alpha}, \vec{R}) \rightarrow \text{RED}_B(\vec{\alpha}, \vec{R})$
- If $T = \forall \gamma. B$, then $\text{RED}_{\forall \gamma. B}(\vec{\alpha}, \vec{R})$ is the set of terms $t$ of type $T\{\vec{\alpha} \backslash \vec{A}\}$ such that for every type $C$ and reducibility candidate $S$ of this type, then $t[C] \in \text{RED}_B(\vec{\alpha} \gamma, \vec{R}S)$
Properties

Lemma

\( \text{RED}_T(\vec{\alpha}, \vec{R}) \) is a reducibility candidate of type \( T\{\vec{\alpha}\backslash \vec{A}\} \)

Lemma

\( \text{RED}_{T\{\gamma\backslash B\}}(\vec{\alpha}, \vec{R}) = \text{RED}_T(\vec{\alpha}\gamma, \vec{R} \text{RED}_B(\vec{\alpha}, \vec{R})) \)

Lemma

If for every type \( B \) and candidate \( S \), \( t\{\gamma\backslash B\} \in \text{RED}_A(\vec{\alpha}\gamma, \vec{R}S) \), then \( \Lambda \gamma t \in \text{RED}_{\forall \gamma, A}(\vec{\alpha}, \vec{R}) \)

Lemma

If \( t \in \text{RED}_{\forall \gamma, A}(\vec{\alpha}, \vec{R}) \), \( t[B] \in \text{RED}_{A\{\gamma\backslash B\}}(\vec{\alpha}, \vec{R}) \) for every type \( B \).
Reducible Terms

A term $t$ of type $A$ is reducible if $t \in \text{RED}_A(\vec{\alpha}, S\vec{N})$ where $\vec{\alpha} = \alpha_1 \ldots \alpha_n$ are the free type variables of $A$, and $S\vec{N}$ is $SN_1 \ldots SN_n$, where $SN_i$ is the set of terms of $SN_F$ of type $\alpha_i$. 
Final Theorem

Theorem

All terms of system \( F \) are reducible.

Corollary (by CR1)

Corollary

All terms of system \( F \) are in \( SN_F \).
Lemma

Let $t$ be a term of type $A$. Suppose $\text{fv}(t) \subseteq \{x_1, \ldots, x_n\}$ and $x_i$ is of type $B_i$. Suppose $t\text{fv}(A, B_1, \ldots, B_n) \subseteq \{\alpha_1, \ldots, \alpha_m\}$. If $\{R_1, \ldots, R_m\}$ are reducibility candidates of types $\{C_1, \ldots, C_m\}$ and $v_1, \ldots, v_n$ are terms of types $B_1\{\vec{\alpha} \setminus \vec{C}\}, \ldots, B_n\{\vec{\alpha} \setminus \vec{C}\}$ which are in $\text{RED}_{B_1}(\vec{\alpha}, \vec{R}), \ldots, \text{RED}_{B_n}(\vec{\alpha}, \vec{R})$, then $t\{\vec{\alpha} \setminus \vec{C}\}\{\vec{x} \setminus \vec{v}\} \in \text{RED}_A(\vec{\alpha}, \vec{R})$