Programming Languages

Unification
Type inference

The type inference problem

Notation

Terms **without** type annotations:

$$U ::= x \mid \lambda x.\, U \mid U\, U \mid \mathsf{True} \mid \mathsf{False} \mid \mathsf{if}\ U\ \mathsf{then}\ U\ \mathsf{else}\ U$$

Terms **with** type annotations:

$$M ::= x \mid \lambda x : \tau.\, M \mid M\, M \mid \mathsf{True} \mid \mathsf{False} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$

We denote by $\mathtt{erase}(M)$ the term without type annotations that results from erasing the type annotations of M.

Example: $\mathtt{erase}((\lambda x : \mathsf{Bool}.\, x)\,\mathsf{True}) = (\lambda x.\, x)\,\mathsf{True}$.

The type inference problem

Definition

A term U without type annotations is **typable** iff there exist:

a typing context Γ

a term with type annotations M

a type τ

such that $\texttt{erase}(M) = U$ and $\Gamma \vdash M : \tau$.

The **type inference problem** consists of:

- Given a term U, determine if it is typable.
- If U is typable:
  find a context Γ, a term M, and a type τ
  such that $\texttt{erase}(M) = U$ and $\Gamma \vdash M : \tau$.

We will see an algorithm to solve this problem.

The type inference problem

The algorithm is based on manipulating *partially known* types.

Example — partially known types

- In $x\,\mathsf{True}$ we know that $x : \mathsf{Bool} \to \mathtt{X}_1$.
- In $\mathsf{if}\ x\,y\ \mathsf{then}\ \mathsf{True}\ \mathsf{else}\ \mathsf{False}$ we know that $x : \mathtt{X}_2 \to \mathsf{Bool}$.

We incorporate *unknowns* $(\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3, \ldots)$ into types.

We will need to solve *equations* between types with unknowns.

Example — equations between types

- $(\mathtt{X}_1 \to \mathsf{Bool}) \stackrel{?}{=} ((\mathsf{Bool} \to \mathsf{Bool}) \to \mathtt{X}_2)$
  has solution: $\mathtt{X}_1 := (\mathsf{Bool} \to \mathsf{Bool})$ and $\mathtt{X}_2 := \mathsf{Bool}$.
- $(\mathtt{X}_1 \to \mathtt{X}_1) \stackrel{?}{=} ((\mathsf{Bool} \to \mathsf{Bool}) \to \mathtt{X}_2)$
  has solution: $\mathtt{X}_1 := (\mathsf{Bool} \to \mathsf{Bool})$ and $\mathtt{X}_2 := (\mathsf{Bool} \to \mathsf{Bool})$.
- $(\mathtt{X}_1 \to \mathsf{Bool}) \stackrel{?}{=} \mathtt{X}_1$
  has no solution.

Unification

We assume a fixed finite set of type constructors:

- Constant types: Bool, Int, ...
- Unary constructors: (List $\bullet$), (Maybe $\bullet$), ...
- Binary constructors: $(\bullet \to \bullet)$, $(\bullet \times \bullet)$, (Either $\bullet$ $\bullet$), ...
- (Etcetera).

Types are formed using unknowns and constructors:

$$\tau ::= \mathtt{X}_n \mid C(\tau_1, \ldots, \tau_n)$$

Unification is the problem of solving systems of equations between types with unknowns.

We will first see a unification algorithm.
Then we will use it to give a type inference algorithm.

Unification

A **substitution** is a function that associates a type with each unknown.

We denote:

$$\{\mathtt{X}_{k_1} := \tau_1, \ldots, \mathtt{X}_{k_n} := \tau_n\}$$

the substitution $\mathbf{S}$ such that $\mathbf{S}(\mathtt{X}_{k_i}) = \tau_i$ for each $1 \leq i \leq n$ and $\mathbf{S}(\mathtt{X}_k) = \mathtt{X}_k$ for any other unknown.

If τ is a type, we write $\mathbf{S}(\tau)$ for the result of replacing each unknown in τ by the value assigned by $\mathbf{S}$.

Example — applying a substitution to a type

If $\mathbf{S} = \{\mathtt{X}_1 := \mathsf{Bool},\ \mathtt{X}_3 := (\mathtt{X}_2 \to \mathtt{X}_2)\}$, then:

$$\mathbf{S}((\mathtt{X}_1 \to \mathsf{Bool}) \to \mathtt{X}_3) = ((\mathsf{Bool} \to \mathsf{Bool}) \to (\mathtt{X}_2 \to \mathtt{X}_2))$$

Unification

A **unification problem** is a finite set E of equations between types that may involve unknowns:

$$E = \{\tau_1 \stackrel{?}{=} \sigma_1, \tau_2 \stackrel{?}{=} \sigma_2, \ldots, \tau_n \stackrel{?}{=} \sigma_n\}$$

A **unifier** for E is a substitution $\mathbf{S}$ such that:

$$\mathbf{S}(\tau_1) = \mathbf{S}(\sigma_1)$$

$$\mathbf{S}(\tau_2) = \mathbf{S}(\sigma_2)$$

$$\ldots$$

$$\mathbf{S}(\tau_n) = \mathbf{S}(\sigma_n)$$

Unification

In general, the solution to a unification problem is not unique.

Example — unification problem with infinite solutions

$$\{X_1 \stackrel{?}{=} X_2\}$$

has infinitely many unifiers:

- $\{X_1 := X_2\}$
- $\{X_2 := X_1\}$
- $\{X_1 := X_3,\ X_2 := X_3\}$
- $\{X_1 := \text{Bool},\ X_2 := \text{Bool}\}$
- $\{X_1 := (\text{Bool} \to \text{Bool}),\ X_2 := (\text{Bool} \to \text{Bool})\}$
- $\dots$

Unification

A substitution $\mathbf{S}_A$ is **more general** than a substitution $\mathbf{S}_B$ if there exists a substitution $\mathbf{S}_C$ such that:

$$\mathbf{S}_B = \mathbf{S}_C \circ \mathbf{S}_A$$

i.e., $\mathbf{S}_B$ is obtained by instantiating variables of $\mathbf{S}_A$.

For the following unification problem:

$$E = \{(\mathtt{X}_1 \to \text{Bool}) \stackrel{?}{=} \mathtt{X}_2\}$$

the following substitutions are unifiers:

- $\mathbf{S}_1 = \{\mathtt{X}_1 := \text{Bool},\ \mathtt{X}_2 := (\text{Bool} \to \text{Bool})\}$
- $\mathbf{S}_2 = \{\mathtt{X}_1 := \text{Int},\ \mathtt{X}_2 := (\text{Int} \to \text{Bool})\}$
- $\mathbf{S}_3 = \{\mathtt{X}_1 := \mathtt{X}_3,\ \mathtt{X}_2 := (\mathtt{X}_3 \to \text{Bool})\}$
- $\mathbf{S}_4 = \{\mathtt{X}_2 := (\mathtt{X}_1 \to \text{Bool})\}$

What is the relationship between them? (Which is more general than which?).

Martelli–Montanari unification algorithm

Given a unification problem E (set of equations):

- While $E \neq \varnothing$, successively apply one of the six rules detailed below.
- The rule may result in `failure`.
- Otherwise, the rule is of the form $E \rightarrow_{\mathbf{S}} E'$.
  The resolution of problem E reduces to solving another problem E', applying the substitution $\mathbf{S}$.

There are two possibilities:

1. $E = E_0 \rightarrow_{\mathbf{S}_1} E_1 \rightarrow_{\mathbf{S}_2} E_2 \rightarrow \ldots \rightarrow_{\mathbf{S}_n} E_n \rightarrow_{\mathbf{S}_{n+1}}$ `failure`
   In that case, the unification problem E has no solution.
2. $E = E_0 \rightarrow_{\mathbf{S}_1} E_1 \rightarrow_{\mathbf{S}_2} E_2 \rightarrow \ldots \rightarrow_{\mathbf{S}_n} E_n = \varnothing$
   In that case, the unification problem E has a solution.

Martelli–Montanari unification algorithm

$$\{\mathtt{X}_n \stackrel{?}{=} \mathtt{X}_n\} \cup E \xrightarrow{\texttt{Delete}} E$$

$$\{C(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C(\sigma_1, \ldots, \sigma_n)\} \cup E \xrightarrow{\texttt{Decompose}} \{\tau_1 \stackrel{?}{=} \sigma_1, \ldots, \tau_n \stackrel{?}{=} \sigma_n\} \cup E$$

$$\{\tau \stackrel{?}{=} \mathtt{X}_n\} \cup E \xrightarrow{\texttt{Swap}} \{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E$$

if τ is not an unknown

$$\{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E \xrightarrow{\texttt{Elim}}_{\{\mathtt{X}_n := \tau\}} E' = \{\mathtt{X}_n := \tau\}(E)$$

if $\mathtt{X}_n$ does not occur in τ

$$\{C(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C'(\sigma_1, \ldots, \sigma_m)\} \cup E \xrightarrow{\texttt{Clash}} \texttt{failure}$$

if $C \neq C'$

$$\{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E \xrightarrow{\texttt{Occurs-Check}} \texttt{failure}$$

if $\mathtt{X}_n \neq \tau$

and $\mathtt{X}_n$ occurs in τ

Martelli–Montanari unification algorithm

Theorem (Correctness of the Martelli–Montanari algorithm)

1. The algorithm terminates for any unification problem E.
2. If E has no solution, the algorithm reaches a `failure`.
3. If E has a solution, the algorithm reaches $\varnothing$:

   $$E = E_0 \rightarrow_{\mathbf{S}_1} E_1 \rightarrow_{\mathbf{S}_2} E_2 \rightarrow \ldots \rightarrow_{\mathbf{S}_n} E_n = \varnothing$$

   Moreover, $\mathbf{S} = \mathbf{S}_n \circ \ldots \circ \mathbf{S}_2 \circ \mathbf{S}_1$ is a unifier for E.

   Moreover, that unifier is the *most general* possible.
   (Up to renaming of unknowns).

Definition (Most general unifier)

We denote $\mathrm{mgu}(E)$ as the most general unifier of E, if it exists.

Martelli–Montanari unification algorithm

Example

Calculate most general unifiers for the following unification problems:

- $\{(\mathtt{X}_2 \rightarrow (\mathtt{X}_1 \rightarrow \mathtt{X}_1)) \stackrel{?}{=} ((\mathsf{Bool} \rightarrow \mathsf{Bool}) \rightarrow (\mathtt{X}_1 \rightarrow \mathtt{X}_2))\}$
- $\{\mathtt{X}_1 \stackrel{?}{=} (\mathtt{X}_2 \rightarrow \mathtt{X}_2),\ \mathtt{X}_2 \stackrel{?}{=} (\mathtt{X}_1 \rightarrow \mathtt{X}_1)\}$

Algorithm $\mathcal{I}$ — Type inference

Algorithm $\mathcal{I}$ receives a term U without type annotations.

It consists of the following steps:

1. **Rectification** of the term.
2. **Annotation** of the term with fresh type variables.
3. **Constraint generation** (equations between types).
4. **Unification** of the constraints.

Algorithm $\mathcal{I}$ — Step 1: rectification

We say a term is *rectified* if:

1. No two bound variables have the same name.
2. No bound variable has the same name as a free variable.

Example – Rectified terms

$x\,(\lambda x.\, x\, x)\,(\lambda y.\, y\, x)$	is not rectified
$x\,(\lambda z.\, z\, z)\,(\lambda y.\, y\, x)$	is rectified
$\lambda x.\, \lambda x.\, x\, y$	is not rectified
$\lambda x.\, \lambda z.\, z\, y$	is rectified

Observation

A term can always be rectified by α-renaming.

Algorithm $\mathcal{I}$ — Step 2: annotation

We have a term U, assumed to be already rectified.

We produce a context Γ_0 and a term M_0:

1. The context Γ_0 assigns types to all free variables of U.
   The type of each variable is a *fresh* unknown.
2. The term M_0 is annotated such that $\texttt{erase}(M_0) = U$.
   All annotations are fresh unknowns.

Example – Annotation of a term

Given the rectified term $U = (\lambda x.\, y\, x\, x)\, (\lambda z.\, w)$, we produce:

1. $\Gamma_0 = (y : \texttt{X}_1, w : \texttt{X}_2)$
2. $M_0 = (\lambda x : \texttt{X}_3.\, y\, x\, x)\, (\lambda z : \texttt{X}_4.\, w)$

Algorithm $\mathcal{I}$ — Step 3: constraint generation

We have a context Γ and a term M with type annotations.

Recursively, we calculate:

1. A type τ, which corresponds to the type of M.
2. A set of equations E.
   They represent constraints for M to be well-typed.

We define a recursive algorithm:

$$\mathcal{I}\,\underbrace{(\,\underbrace{\Gamma}_{\text{context}} \mid \underbrace{M}_{\text{term}}\,)}_{\text{input}} = \underbrace{(\,\underbrace{\tau}_{\text{type}} \mid \underbrace{E}_{\text{constraints}}\,)}_{\text{output}}$$

with the precondition that Γ assigns types to all variables of M.

Algorithm $\mathcal{I}$ — Step 3: constraint generation

1. $\mathcal{I}(\Gamma \mid \text{True}) = (\text{Bool} \mid \varnothing)$
2. $\mathcal{I}(\Gamma \mid \text{False}) = (\text{Bool} \mid \varnothing)$
3. $\mathcal{I}(\Gamma \mid x) = (\tau \mid \varnothing) \qquad \text{if } (x : \tau) \in \Gamma$
4. $\mathcal{I}(\Gamma \mid \text{if } M_1 \text{ then } M_2 \text{ else } M_3) =$
   $$(\tau_2 \mid \{\tau_1 \stackrel{?}{=} \text{Bool}, \tau_2 \stackrel{?}{=} \tau_3\} \cup E_1 \cup E_2 \cup E_3)$$
   where $\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1)$
   $\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2)$
   $\mathcal{I}(\Gamma \mid M_3) = (\tau_3 \mid E_3)$
5. $\mathcal{I}(\Gamma \mid M_1\ M_2) = (\mathtt{X}_k \mid \{\tau_1 \stackrel{?}{=} (\tau_2 \to \mathtt{X}_k)\} \cup E_1 \cup E_2)$
   where $\mathtt{X}_k$ is a fresh unknown
   $\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1)$
   $\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2)$
6. $\mathcal{I}(\Gamma \mid \lambda x : \tau.\ M) = (\tau \to \sigma \mid E)$
   where $\mathcal{I}(\Gamma, x : \tau \mid M) = (\sigma \mid E)$

Algorithm $\mathcal{I}$ — Step 4: unification of constraints

Recall: Γ_0 and M_0 result from annotating a rectified term U.

Once we have calculated $\mathcal{I}(\Gamma_0 \mid M_0) = (\tau \mid E)$:

1. Calculate $\mathbf{S} = \mathrm{mgu}(E)$.
2. If the unifier does not exist, the term U is not typable.
3. If the unifier exists, the term U is typable and we have:

$$\mathbf{S}(\Gamma_0) \vdash \mathbf{S}(M_0) : \mathbf{S}(\tau)$$

Algorithm $\mathcal{I}$ — Correctness

Theorem (Correctness of algorithm $\mathcal{I}$)

Let Γ_0 and M_0 be the result of annotating a rectified term U. Assume that $\mathcal{I}(\Gamma_0 \mid M_0) = (\tau \mid E)$. Then:

1. If U is not typable, there is no unifier for E.
2. If U is typable, there exists $\mathbf{S} = \text{mgu}(E)$.
   Moreover, $\mathbf{S}(\Gamma_0) \vdash \mathbf{S}(M_0) : \mathbf{S}(\tau)$ is a valid typing judgment.

   Furthermore, this typing judgment is the most general possible for U.
   More precisely, if $\Gamma' \vdash M' : \tau'$ is a valid judgment and $\texttt{erase}(M') = U$, there exists a substitution $\mathbf{S}'$ such that:

$$\begin{array}{rcl} \Gamma' & \supseteq & \mathbf{S}'(\Gamma_0) \\ M' & = & \mathbf{S}'(M_0) \\ \tau' & = & \mathbf{S}'(\tau) \end{array}$$

   where additionally $\mathbf{S}$ is more general than $\mathbf{S}'$.

Algorithm $\mathcal{I}$ for type inference

Exercise. Apply the inference algorithm to the following terms:

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

Recall: unification algorithm

$$\{\mathtt{X}_n \stackrel{?}{=} \mathtt{X}_n\} \cup E \xrightarrow{\texttt{Delete}} E$$

$$\{C(\tau_1,\ldots,\tau_n) \stackrel{?}{=} C(\sigma_1,\ldots,\sigma_n)\} \cup E \xrightarrow{\texttt{Decompose}} \{\tau_1 \stackrel{?}{=} \sigma_1,\ldots,\tau_n \stackrel{?}{=} \sigma_n\} \cup E$$

$$\{\tau \stackrel{?}{=} \mathtt{X}_n\} \cup E \xrightarrow{\texttt{Swap}} \{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E$$

if τ is not an unknown

$$\{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E \xrightarrow{\texttt{Elim}}_{\{\mathtt{X}_n := \tau\}} E\{\mathtt{X}_n := \tau\}$$

if $\mathtt{X}_n$ does not occur in τ

$$\{C(\tau_1,\ldots,\tau_n) \stackrel{?}{=} C'(\sigma_1,\ldots,\sigma_m)\} \cup E \xrightarrow{\texttt{Clash}} \texttt{failure}$$

if $C \neq C'$

$$\{\mathtt{X}_n \stackrel{?}{=} \tau\} \cup E \xrightarrow{\texttt{Occurs-Check}} \texttt{failure}$$

if $\mathtt{X}_n \neq \tau$ and $\mathtt{X}_n$ occurs in τ

Termination of the unification algorithm

Given a set of unification equations E, we define:

- n_1: number of distinct unknowns in E
- n_2: size of E, calculated as $\sum_{(\tau \stackrel{?}{=} \sigma) \in E} |\tau| + |\sigma|$
- n_3: number of equations of the form $\tau \stackrel{?}{=} \mathtt{X}_n$ in E

We can observe that the rules that do not produce failure decrease the triple (n_1, n_2, n_3), according to the *lexicographic order*:

	n_1	n_2	n_3
`Elim`	$>$		
`Decompose`	$=$	$>$	
`Delete`	$\geq$	$>$	
`Swap`	$=$	$=$	$>$

Correctness of the unification algorithm

Recall

1. A **substitution** is a function **S**
   that associates a type $\mathbf{S}(\mathtt{X}_n)$ with each unknown $\mathtt{X}_n$.
2. **S** is a **unifier** of E
   if for each $(\tau \stackrel{?}{=} \sigma) \in E$ we have $\mathbf{S}(\tau) = \mathbf{S}(\sigma)$.
3. **S** is **more general** than $\mathbf{S}'$
   if there exists **T** such that $\mathbf{S}' = \mathbf{T} \circ \mathbf{S}$.
4. **S** is an **m.g.u.** of E if **S** is a unifier of E
   and for every unifier $\mathbf{S}'$ of E
   we have that **S** is more general than $\mathbf{S}'$.

   Technically, we are interested in **idempotent** m.g.u.'s,
   i.e., $\mathbf{S}(\mathbf{S}(\tau)) = \mathbf{S}(\tau)$ for every term τ.

Correctness of the unification algorithm

Lemma — correctness of the Delete rule

$\mathbf{S}$ m.g.u. of $E \implies \mathbf{S}$ m.g.u. of $\{\mathtt{X}_n \stackrel{?}{=} \mathtt{X}_n\} \cup E$.

Lemma — correctness of the Swap rule

$\mathbf{S}$ m.g.u. of $\{\tau \stackrel{?}{=} \sigma\} \cup E \implies \mathbf{S}$ m.g.u. of $\{\sigma \stackrel{?}{=} \tau\} \cup E$.

Lemma — correctness of the Decompose rule

$\mathbf{S}$ m.g.u. of $\{\tau_1 \stackrel{?}{=} \sigma_1, \ldots, \tau_n \stackrel{?}{=} \sigma_n\} \cup E$

$\implies \mathbf{S}$ m.g.u. of $\{C(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C(\sigma_1, \ldots, \sigma_n)\} \cup E$.

Lemma — correctness of the Elim rule

$\mathbf{S}$ m.g.u. of $E\{\mathtt{X}_n := \tau\}$ and $\mathtt{X}_n \notin \tau$

$\implies \mathbf{S} \circ \{\mathtt{X}_n := \tau\}$ m.g.u. of E.

Use that if $\mathbf{S}(\mathtt{X}_n) = \tau$ then $\mathbf{S}(\sigma\{\mathtt{X}_n := \tau\}) = \mathbf{S}(\sigma)$.

Correctness of the unification algorithm

Let's prove correctness in case of success.

Let $E_0 \to_{\mathbf{S}_1} E_1 \to_{\mathbf{S}_n} E_2 \to \ldots \to_{\mathbf{S}_n} E_n = \varnothing$.
We need to show that $\mathbf{S}_n \circ \ldots \circ \mathbf{S}_1$ is an m.g.u. of E.
By induction on n:

1. If $n = 0$, the identity substitution is an m.g.u. of $\varnothing$.
2. If $n > 0$, we have:

$$E_0 \to_{\mathbf{S}_1} E_1 \qquad E_1 \to_{\mathbf{S}_2} \ldots \to_{\mathbf{S}_n} E_n = \varnothing$$

By IH, $\mathbf{S}_n \circ \ldots \circ \mathbf{S}_2$ is an m.g.u. of E_1.
Applying one of the previous lemmas, we conclude that
$\mathbf{S}_n \circ \ldots \circ \mathbf{S}_2 \circ \mathbf{S}_1$ is an m.g.u. of E_0.

Correctness of the unification algorithm

Correctness in case of failure is proved similarly,
with lemmas that go "forward" instead of "backward".

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ? ? ? ? ? ? ? ? ?

Recommended reading

Chapter 22 of Pierce's book.
Benjamin C. Pierce. *Types and Programming Languages*.
The MIT Press, 2002.

Extra: theory behind the unification method

Section 4.5 of the book by Baader & Nipkow.
Franz Baader and Tobias Nipkow. *Term Rewriting and All That*.
Cambridge University Press, 1998.