Type Reconstruction

Advanced Compiler Construction and Program Analysis

Lecture 11

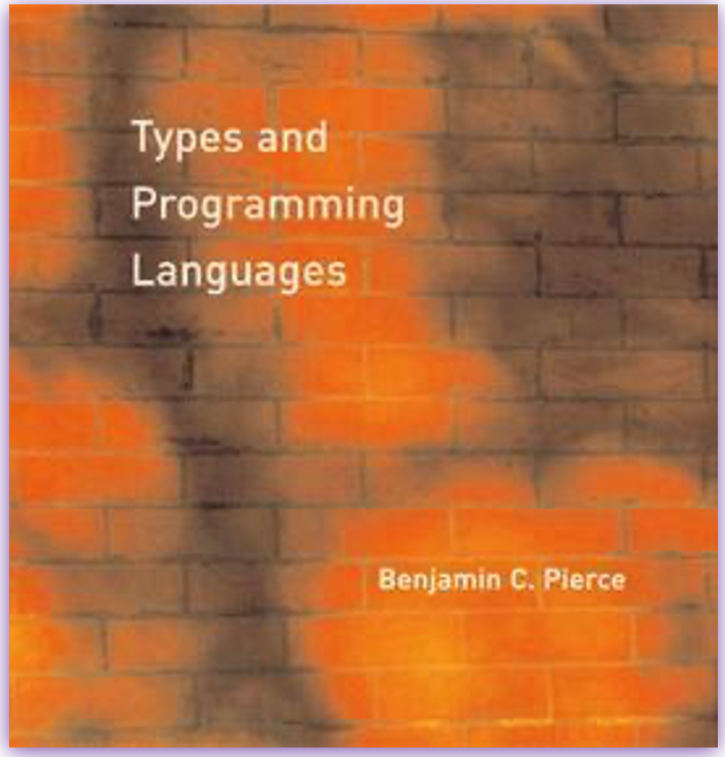
Innopolis University, Spring 2022

The topics of this lecture are covered in detail in…

Benjamin C. Pierce.

Types and Programming Languages

MIT Press 2002



Type Variables: substitution

Example: $\sigma = [X \mapsto \mathtt{Bool},\ Z \mapsto \mathtt{Nat}\to\mathtt{Nat}]$

$$
\begin{array}{lll}
\sigma(Y) & = T & \text{if } (Y \mapsto T) \in \sigma \\
\sigma(Y) & = Y & \text{otherwise} \\
\sigma(\mathtt{Bool}) & = \mathtt{Bool} & \\
\sigma(\mathtt{Nat}) & = \mathtt{Nat} & \\
\sigma(T_1 \to T_2) & = \sigma(T_1) \to \sigma(T_2) &
\end{array}
$$

Preservation under type substitution

Theorem 11.1. If $\Gamma \vdash t : T$, then $\sigma\Gamma \vdash \sigma t : \sigma T$.

Proof. Straightforward by induction on typing derivations.

Type Variables: two views

Suppose we have a term with type variables in some context. There are at least two different interpretations we might be interested in:

1. Are **all** substitution instances of the term well-typed?
2. Is there **some** well-typed substitution instance?

Type Variables for Parametric Polymorphism

1. Are **all** substitution instances of the term well-typed?

Type Variables for Parametric Polymorphism

1. Are **all** substitution instances of the term well-typed?

`λf:X→X. λa:X. f (f a)` — well-typed for all `X`

Type Variables for Parametric Polymorphism

1. Are **all** substitution instances of the term well-typed?

`λf:X→X. λa:X. f (f a)` — well-typed for all X

Treat type variables
like in Java's generics,
or in like in Haskell

Type Variables for Parametric Polymorphism

1. Are **all** substitution instances of the term well-typed?

`λf:X→X. λa:X. f (f a)` — well-typed for all `X`

```
λf:Bool→Bool. λa:Bool. f (f a)
λf:Nat→Nat.   λa:Nat.  f (f a)
```

Type Variables for Type Reconstruction

2. Is there **some** well-typed substitution instance?

Type Variables for Type Reconstruction

2. Is there **some** well-typed substitution instance?

`λf:X. λa:Y. f (f a)` — ill-typed on its own

Type Variables for Type Reconstruction

2. Is there **some** well-typed substitution instance?

`λf:X. λa:Y. f (f a)` — ill-typed on its own

Treat type variables
like unknowns,
that we need to solve for

Type Variables for Type Reconstruction

2. Is there **some** well-typed substitution instance?

`λf:X. λa:Y. f (f a)` — ill-typed on its own

`[X ↦ Nat→Nat, Y ↦ Nat]`
`λf:Nat→Nat. λa:Nat. f (f a)`

Constraint-Based Typing: intuition

Instead of type *checking*, record ***constraints*** (equations of types), and then try to solve them.

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\ t_2 : X \mid T_1 = T_2 \to X}$$

Constraint Typing Relation

$$\Gamma \vdash t : T \mid C$$

$$\Gamma, x{:}T \vdash x : T \mid \emptyset$$

$$\frac{\Gamma, x{:}T_1 \vdash t : T_2 \mid C}{\Gamma \vdash \lambda x{:}T_1.t : T_1 \to T_2 \mid C}$$

Constraint Typing Relation

$$\Gamma \vdash t : T \mid C$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid C_1 \qquad \Gamma \vdash t_2 : T_2 \mid C_2 \qquad X \text{ is a } \textit{fresh} \text{ type variable}}{\Gamma \vdash t_1\ t_2 : X \mid C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}}$$

Constraint Typing Relation

$$\Gamma \vdash t : T \mid C$$

$$\frac{\Gamma \vdash t : T \mid C}{\Gamma \vdash \textbf{succ}\ t : \mathsf{Nat} \mid C \cup \{T = \mathsf{Nat}\}}$$

$$\Gamma \vdash 0 : \mathsf{Nat} \mid \emptyset$$

$$\frac{\Gamma \vdash t : T \mid C}{\Gamma \vdash \textbf{pred}\ t : \mathsf{Nat} \mid C \cup \{T = \mathsf{Nat}\}}$$

$$\frac{\Gamma \vdash t : T \mid C}{\Gamma \vdash \textbf{iszero}\ t : \mathsf{Bool} \mid C \cup \{T = \mathsf{Nat}\}}$$

Constraint Typing Relation

$$\Gamma \vdash t : T \mid C$$

$$\Gamma \vdash \texttt{false} : \texttt{Bool} \mid \emptyset$$

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \mid \emptyset$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : T_1 \mid C_1 \\ \Gamma \vdash t_2 : T_2 \mid C_2 \\ \Gamma \vdash t_3 : T_3 \mid C_3\end{array}}{\begin{array}{l}\Gamma \vdash \textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 : T_2 \\ \quad \mid C_1 \cup C_2 \cup C_3 \cup \{T_1 = \texttt{Bool}, T_2 = T_3\}\end{array}}$$

Solution to constraint typing

Definition 11.2. A ***solution*** for constraint typing

$$\Gamma \vdash t : S \mid C$$

is a pair (σ, T) such that σ satisfies C and $\sigma S = T$.

Unifiers

Definition 11.3. A substitution σ is ***less specific*** than substitution σ' (written $\sigma \sqsubseteq \sigma'$) if $\sigma' = \sigma;\gamma$ for some substitution γ.

Definition 11.4. A substitution σ is called a ***principal unifier*** for a constraint set C if σ satisfies C and, for any other substitution σ' that satisfies C, we have $\sigma \sqsubseteq \sigma'$.

Unification: exercise

Exercise 11.5. Find a principal unifier for each of the following sets of constraints:

1. {X = Nat, Y = X→X}
2. {X→Y = Y→Z, Z = U→W}
3. {Y = Nat→Y}
4. {Nat→Nat = X→Y}
5. {Nat = Nat→Y}
6. {}

Unification

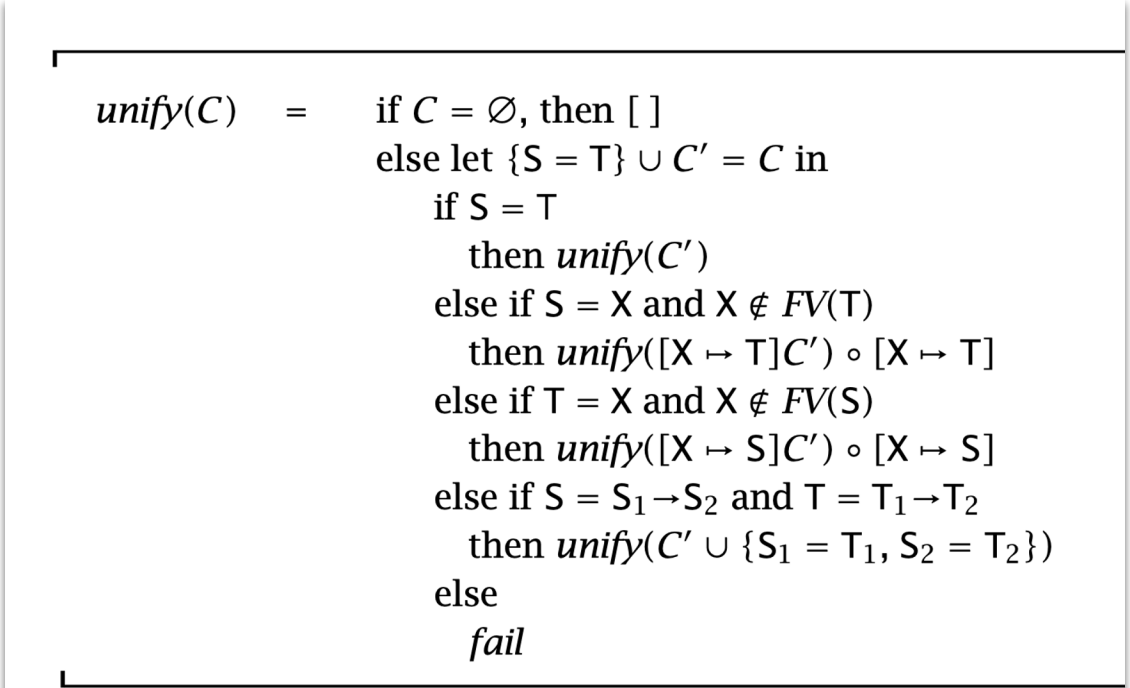


Figure 22-2: Unification algorithm

Unification: termination

Theorem 11.6.

1. `unify(C)` halts, either by failing or by returning a substitution, for all `C`;

2. if `unify(C)` $= \sigma$, then σ is a unifier for `C`;

3. if δ is a unifier for `C`, then `unify(C)`$=\sigma$ with $\sigma \sqsubseteq \delta$.

Principal Types

Definition 11.7. Suppose, a solution for constraint typing `Γ ⊢ t : S | C` is a pair `(σ, T)`.
Then `T` is called a ***principal type*** if for any other solution `(σ', T)`, we have `σ ⊑ σ'`.

Principal Types: exercise

Exercise 11.8. Find a principal type for:

```
λx:X. λy:Y. λz:Z. (x z) (y z)
```

Implicit Type Annotations

$$\frac{X \text{ is a fresh type variable} \qquad \Gamma, x{:}X \vdash t : T_2 \mid C}{\Gamma \vdash \lambda x.t : X \to T_2 \mid C}$$

Let-Polymorphism: motivation

```
let twice = λf:Nat→Nat. λa:Nat. f(f(a))
 in twice (λx:Nat. succ (succ x)) 2

let twice = λf:Bool→Bool. λa:Bool. f(f(a))
 in twice (λx:Bool. x) false
```

Let-Polymorphism

```
let twice = λf. λa. f(f(a)) in
let a = twice (λx:Nat. succ (succ x)) 2 in
let b = twice (λx:Bool. x) false in
if b then a else 0
```

$$\frac{\Gamma \vdash t_1 : T_1 \mid C_1 \qquad \Gamma \vdash [x \mapsto t_1]t_2 : T_2 \mid C_2}{\Gamma \vdash \textbf{let } x = t_1 \textbf{ in } t_2 : T_2 \mid C_1 \cup C_2}$$

Other techniques

Constraint-based type reconstruction is fairly widespread for relatively simple type systems. However, for some type systems, most general unifier might not exist or unification is inefficient.

In those situations, ***bidirectional type checking*** is commonly applied. Also, in presence of type-level computation,
normalization by evaluation is often used to compare types.

Summary

- ❏ Type variables
- ❏ Type reconstruction
- ❏ Constraint-based typing
- ❏ Unification

See you next time!