

CS421 Topic 7: Type Derivations¹

Sameer Sundresh
sundresh@uiuc.edu

University of Illinois at Urbana-Champaign

June 5, 2007

¹Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Mark Hills

Objectives

- ▶ type rules and how to structure a proof-tree
- ▶ how to use the type rules to check the type of an expression
- ▶ how to use the type rules to infer the type of an expression
- ▶ how to write your own type rule for an expression

Format of a Type Judgment

An *type judgment* has the following form:

$$\Gamma \vdash e : \tau$$

where

- ▶ Γ is a *type environment*, which can be seen as a list or set of pairs $x : \tau$ with x a variable or function name and τ the type;
- ▶ e is a program expression;
- ▶ and τ is the *type* to be assigned to e .

Note: the \vdash is pronounced “turnstile”, “entails”, or sometimes “satisfies”.

Examples of Valid Type Judgments

- ▶ $\vdash \text{true} \ \&\& \ \text{false} : \text{bool}$
- ▶ $[x : \text{int}] \vdash x + 3 : \text{int}$
- ▶ $[f : \text{int} \rightarrow \text{string}] \vdash f(5) : \text{string}$

Objectives

- ▶ type rules and how to structure a proof-tree
- ▶ how to use the type rules to check the type of an expression
- ▶ how to use the type rules to infer the type of an expression
- ▶ how to write your own type rule for an expression

Format of a Type Judgment

An *type judgment* has the following form:

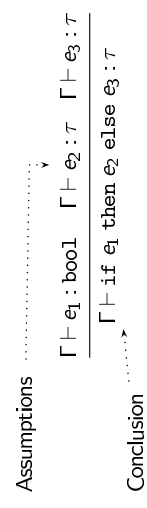
$$\Gamma \vdash e : \tau$$

where

- ▶ Γ is a *type environment*, which can be seen as a list or set of pairs $x : \tau$ with x a variable or function name and τ the type;
- ▶ e is a program expression;
- ▶ and τ is the *type* to be assigned to e .

Note: the \vdash is pronounced “turnstile”, “entails”, or sometimes “satisfies”.

Format of Typing Rules



- ▶ **Core Idea:** Type of expression determined by type of components
- ▶ If a rule has no assumptions, then it is called an *axiom*
- ▶ Γ may be left out if we don't need a type environment
- ▶ Γ , e , and τ are *parameterized* – they may contain *meta-variables* used for typing

Objectives

- ▶ type rules and how to structure a proof-tree
- ▶ how to use the type rules to check the type of an expression
- ▶ how to use the type rules to infer the type of an expression
- ▶ how to write your own type rule for an expression

Format of a Type Judgment

An *type judgment* has the following form:

$$\Gamma \vdash e : \tau$$

where

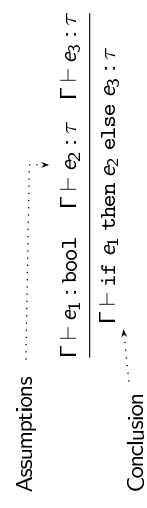
- ▶ Γ is a *type environment*, which can be seen as a list or set of pairs $x : \tau$ with x a variable or function name and τ the type;
- ▶ e is a program expression;
- ▶ and τ is the *type* to be assigned to e .

Note: the \vdash is pronounced “turnstile”, “entails”, or sometimes “satisfies”.

Examples of Valid Type Judgments

- ▶ $\vdash \text{true} \ \&\& \ \text{false} : \text{bool}$
- ▶ $[x : \text{int}] \vdash x + 3 : \text{int}$
- ▶ $[f : \text{int} \rightarrow \text{string}] \vdash f(5) : \text{string}$

Format of Typing Rules



- ▶ **Core Idea:** Type of expression determined by type of components
- ▶ If a rule has no assumptions, then it is called an *axiom*
- ▶ Γ may be left out if we don't need a type environment
- ▶ Γ , e , and τ are *parameterized* – they may contain *meta-variables* used for typing

Axioms – Constants

$$\frac{}{\vdash n : \text{int}} \text{ (assuming } n \text{ is an int.)}$$

$$\frac{}{\vdash \text{true} : \text{bool}}$$

$$\frac{}{\vdash \text{false} : \text{bool}}$$

- ▶ These are rules that are true no matter what the typing context
- ▶ n is a *meta-variable* representing any int

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Axioms – Variables</h2>	

There are many ways you may see this rule formatted. If Γ is a set, it may be shown as:

$$\frac{}{\Gamma \vdash x : \tau} \text{ if } (x : \tau) \in \Gamma$$

It may also be shown as:

$$x : \tau \in \Gamma$$

$$\frac{}{\Gamma \vdash x : \tau}$$

If Γ is treated as a list, order is important, and we will say

$\Gamma(x) = \tau$ if $x : \tau \in \Gamma$ and no other definition of x is to the left of $x : \tau$. Then, we will have:

$$\frac{}{\Gamma \vdash x : \tau} \text{ if } \Gamma(x) = \tau$$

We will use the first.

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Rules</h2>	

► Rules for arithmetic operators (+,*,etc):

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

► Rules for relational operators (=,<,etc) are similar:

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Rules</h2>	

Booleans look similar, except for not, which has only one operand:

$$\frac{\Gamma \vdash e_1 : \text{bool}}{\Gamma \vdash e_1 \ \&\& \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ || \ e_2 : \text{bool}}$$

$$\frac{}{\Gamma \vdash e_1 : \text{bool}}$$

$$\frac{}{\Gamma \vdash ! \ e_1 : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Example</h2>	

Suppose we want to prove that $\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

First thing: Write down the thing you are trying to prove, and put a bar over it. Proofs are from the bottom up.

$$\frac{}{\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Example</h2>	

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Example</h2>	

Suppose we want to prove that $\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Next, look at the *outermost* expression. What kind of expression is this? What rule has this as its conclusion?

$$\frac{}{\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
<h2 style="color: white;">Simple Example</h2>	

Suppose we want to prove that $\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Next, look at the *outermost* expression. What kind of expression is this? What rule has this as its conclusion?

$$\frac{}{\Gamma \vdash y \ || \ (x + 3 > 6) : \text{bool}}$$

Boolean or:

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ || \ e_2 : \text{bool}}$$

This will tell us what we need to do next.

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Functions Let and Letrec
---	---

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Write parts on top and put a bar over them as well.

$$\frac{?}{\Gamma \vdash y : \text{bool}}$$

$$\frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Now, pick an assumption. We can work left to right. Which rule has $\Gamma \vdash y : \text{bool}$ as a conclusion?

$$\frac{?}{\Gamma \vdash y : \text{bool}}$$

$$\frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the rule for the $>$ relational operator. Again, we expand this out.

$$\frac{?}{\Gamma \vdash x + 3 : \text{int}}$$

$$\frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the variable axiom. Now, what rule has $\Gamma \vdash x + 3 > 6 : \text{bool}$ as a conclusion?

$$\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Now, we need to select an assumption again. Which rule has $\Gamma \vdash 6 : \text{int}$ as its conclusion?

$$\frac{?}{\Gamma \vdash x + 3 : \text{int}}$$

$$\frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the axiom for constants. Now, we can select the other assumption. Which rule has $\Gamma \vdash x + 3 : \text{int}$ as a conclusion?

$$\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the rule for arithmetic operations.

$$\frac{?}{\Gamma \vdash x : \text{int}} \quad \frac{?}{\Gamma \vdash 3 : \text{int}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Sameer Sundresh CS421, Topic 7: Type Derivations

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

Now, we pick an assumption to prove again. Which rule has $\Gamma \vdash 3 : \text{int}$ as a conclusion?

$$\frac{?}{\Gamma \vdash x : \text{int}} \quad \frac{?}{\Gamma \vdash 3 : \text{int}} \quad \frac{\Gamma \vdash x + 3 : \text{int}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Sameer Sundresh CS421, Topic 7: Type Derivations

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the axiom for constants. Which rule has $\Gamma \vdash x : \text{int}$ as a conclusion?

$$\frac{?}{\Gamma \vdash x : \text{int}} \quad \frac{?}{\Gamma \vdash 3 : \text{int}} \quad \frac{\Gamma \vdash x + 3 : \text{int}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Sameer Sundresh CS421, Topic 7: Type Derivations

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Example Type Variables Functions Let and Letrec
---	---

Simple Example

Suppose we want to prove that $\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}$.
 Assume that $\Gamma = [x : \text{int} ; y : \text{bool}]$

This is the rule for variables. Now, we have no more assumptions, and our proof is **DONE!**

$$\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}{\Gamma \vdash y \mid \mid (x + 3 > 6) : \text{bool}}$$

Sameer Sundresh CS421, Topic 7: Type Derivations

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Type Variables Functions Let and Letrec
---	--

Type Variables in Rules

$$\text{If } \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}$$

- ▶ τ is a *type variable*, or *meta-variable*
- ▶ It means “any type at all”—but whatever type τ you pick it has to be the same for the three places it shows up in this rule.
- ▶ So... the `if` rule says that `if` can result in any type, as long as the `then` and `else` branches have the same type. This could even include functions.

Sameer Sundresh CS421, Topic 7: Type Derivations

Objectives Type Judgments Typing Rules Polymorphic Functions Extended Example Wrap-Up	General Format Axioms Simple Rules Type Variables Functions Let and Letrec
---	--

Function Application

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

- ▶ If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$, applied to an argument expression e_2 of type τ_1 , the resulting expression has type τ_2

Sameer Sundresh CS421, Topic 7: Type Derivations

Function Application Example

$$\frac{}{\Gamma \vdash \text{print_int} : \text{int} \rightarrow \text{unit} \quad \Gamma \vdash 5 : \text{int}}$$

- ▶ $e_1 = \text{print_int}$
- ▶ $e_2 = 5$
- ▶ $\tau_1 = \text{int}$
- ▶ $\tau_2 = \text{unit}$.

$$\frac{}{\Gamma \vdash \text{print_int } 5 : \text{unit}}$$

- ▶ $e_1 = \text{print_int}$
- ▶ $e_2 = 5$
- ▶ $\tau_1 = \text{int}$
- ▶ $\tau_2 = \text{unit}$.

Function Application Example 2

$$\frac{}{\Gamma \vdash \text{map print_int} : \text{int list} \rightarrow \text{unit list} \quad \Gamma \vdash [3;7] : \text{int list}}$$

- ▶ $e_1 = \text{map print_int}$
- ▶ $e_2 = [3;7]$
- ▶ $\tau_1 = \text{int list}$
- ▶ $\tau_2 = \text{unit list}$

$$\frac{}{\Gamma \vdash \text{map print_int } [3;7] : \text{unit list}}$$

- ▶ $e_1 = \text{map print_int}$
- ▶ $e_2 = [3;7]$
- ▶ $\tau_1 = \text{int list}$
- ▶ $\tau_2 = \text{unit list}$

Functions

- ▶ Important point: the rules describe types, but they also describe when you may change Γ .
- ▶ You may **NOT** change Γ except as described!

The rule for functions adds the bound variable to the type environment:

$$\frac{}{\Gamma \cup [x : \tau_1] \vdash e : \tau_2}$$

$$\frac{}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

Function Examples

$$\frac{}{\Gamma \cup [y : \text{int}] \vdash y + 3 : \text{int}}$$

$$\frac{}{\Gamma \vdash \text{fun } y \rightarrow y + 3 : \text{int} \rightarrow \text{int}}$$

$$\frac{}{\Gamma \cup [f : \text{int} \rightarrow \text{bool}] \vdash (f 2) :: [\text{true}] : \text{bool list}}$$

$$\frac{}{\Gamma \vdash (\text{fun } f \rightarrow (f 2)) :: [\text{true}] : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool list}}$$

Let and Letrec

- ▶ Let Rule:

$$\frac{}{\Gamma \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}$$
- ▶ Letrec Rule:

$$\frac{}{\Gamma \cup [x : \tau] \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}$$

$$\frac{}{\Gamma \vdash \text{let } \text{rec } x = e_1 \text{ in } e_2 : \tau'}$$

Some caveats...

The above system cannot handle polymorphism like we find in OCaml. Specifically, we have meta-variables in our proof logic, but no type variables in the type language. To handle this properly, we would need:

- ▶ Object level type variables
- ▶ Type quantification (universal types)
- ▶ **let** and **letrec** rules to introduce polymorphism
- ▶ Explicit rules to eliminate (instantiate) polymorphism

Some caveats...

The above system cannot handle polymorphism like we find in OCaml. Specifically, we have meta-variables in our proof logic, but no type variables in the type language. To handle this properly, we would need:

- ▶ Object level type variables
- ▶ Type quantification (universal types)
- ▶ **let** and **letrec** rules to introduce polymorphism
- ▶ Explicit rules to eliminate (instantiate) polymorphism

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphism

Polymorphism = many forms—a polymorphic function can take an argument of several different types. Two kinds:

- ▶ **parametric polymorphism**—all types must be treated uniformly
- ▶ **ad hoc polymorphism**—can have different behavior on data of different types

OCaml provides *parametric polymorphism* using the **Hindley-Damas-Milner** type system (which also allows **type inference**—to be discussed next week). Ad hoc polymorphism can be simulated with variants (discussed yesterday).

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Type Schemas

Just like a plain (*monomorphic*) function has a *type*, a *polymorphic* function has a **type schema**.

$$\forall\{\alpha, \dots\}. \tau \rightarrow \tau'$$

Where τ is a plain type that may involve the **type variables** in $\{\alpha, \dots\}$. We will also denote a (possibly empty) set of type variables as A . For example,

$$\vdash \text{fun } x \rightarrow x : \forall\{ 'a \}. 'a \rightarrow 'a$$

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Types as Type Schemas

A plain type is just a type schema without any type variables.

$$\tau = \forall\{ \}. \tau$$

This just makes our typing rules a little easier because we don't always have to distinguish between types and type schemas.

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphic Function Example

Here's how we would compute the type of a polymorphic function.

$$\frac{\dots}{\Gamma \cup [x : 'a, y : 'b] \vdash (x, y) : 'a * 'b}$$

$$\frac{\Gamma \cup [x : 'a] \vdash \text{fun } y \rightarrow (x, y) : \forall 'b. 'b \rightarrow 'a * 'b}{\Gamma \vdash \text{fun } x \rightarrow \text{fun } y \rightarrow (x, y) : \forall\{ 'a, 'b \}. 'a \rightarrow 'b \rightarrow 'a * 'b}$$

Type variables are only quantified if they do not appear in Γ .

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphic Function Application

This is the typing rule used when applying a polymorphic function to an argument:

$$\frac{\Gamma \vdash f : \forall\{\alpha, \beta, \dots\}. \tau \rightarrow \tau' \quad \Gamma \vdash x : \tau[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]}{\Gamma \vdash f \ x : \tau'[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]}$$

Where $\tau[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]$ means τ with type variable α replaced by type expression τ_1 , type variable β replaced by τ_2 , etc. In other words, the type of x is an **instantiation** of the polymorphic argument type of f .

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphic Functions

This is the typing rule for a polymorphic function:

$$\frac{\Gamma \cup [x : \tau_1] \vdash e : \forall A. \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \forall \nu(\Gamma, \tau_1) \cup A. \tau_1 \rightarrow \tau_2}$$

Where $\nu(\Gamma, \tau_1)$ is the set of **free type variables** in τ_1 , i.e. the type variables in τ_1 that are not mentioned in Γ .

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphic Function Example

Here's how we would compute the type of a polymorphic function.

$$\frac{\dots}{\Gamma \cup [x : 'a, y : 'b] \vdash (x, y) : 'a * 'b}$$

$$\frac{\Gamma \cup [x : 'a] \vdash \text{fun } y \rightarrow (x, y) : \forall 'b. 'b \rightarrow 'a * 'b}{\Gamma \vdash \text{fun } x \rightarrow \text{fun } y \rightarrow (x, y) : \forall\{ 'a, 'b \}. 'a \rightarrow 'b \rightarrow 'a * 'b}$$

Type variables are only quantified if they do not appear in Γ .

Objectives
Type Judgments
Typing Rules
Polymorphic Functions
Extended Example
Wrap-Up

Polymorphism
Type Schemas
Polymorphic Functions
Polymorphic Function Application
Polymorphic Let and Letrec

Polymorphic Function Application

This is the typing rule used when applying a polymorphic function to an argument:

$$\frac{\Gamma \vdash f : \forall\{\alpha, \beta, \dots\}. \tau \rightarrow \tau' \quad \Gamma \vdash x : \tau[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]}{\Gamma \vdash f \ x : \tau'[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]}$$

Where $\tau[\alpha \leftarrow \tau_1, \beta \leftarrow \tau_2, \dots]$ means τ with type variable α replaced by type expression τ_1 , type variable β replaced by τ_2 , etc. In other words, the type of x is an **instantiation** of the polymorphic argument type of f .

Type Instantiations

A type schema involving quantified type variables may be **instantiated** to another type schema or a plain type.

- Simply replace some or all of the quantified type variables with type expressions.

Examples:

- $\forall\{a \text{ list}\}. 'a \text{ list} \rightarrow 'a \Rightarrow$
 $\text{int list} \rightarrow \text{int}, \text{ string list} \rightarrow \text{string}, \dots$
- $\forall\{a, 'b\}. 'a \rightarrow 'b \rightarrow 'a * 'b \Rightarrow$
 $\text{int} \rightarrow \text{string} \rightarrow \text{int} * \text{string}$
- $\forall\{a\}. 'a \rightarrow 'a \Rightarrow \forall\{b\}. ('b \rightarrow 'b) \rightarrow ('b \rightarrow 'b)$

Polymorphic Let and Letrec

We need to slightly change the typing rules for Let and Letrec to accommodate polymorphic functions, which have *type schemas* rather than plain *types*.

- Let Rule:
$$\frac{\Gamma \vdash e_1 : \forall A.\tau \quad \Gamma \cup [x : \forall A.\tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$
- Letrec Rule:
$$\frac{\Gamma \cup [x : \forall A.\tau] \vdash e_1 : \forall A.\tau \quad \Gamma \cup [x : \forall A.\tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x = e_1 \text{ in } e_2 : \tau'}$$

A more advanced example

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in}$
 $\text{let } x = 2 \text{ in}$
 $\text{fun } y \rightarrow \lambda(x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow$
 int list

What rule has this as the conclusion?

?

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow$
 int list

This uses the letrec rule.

#1 #2

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

#1 = $\Gamma \cup [\text{one} : \text{int list}] \vdash (1 :: \text{one}) : \text{int list}$

#2 = $\Gamma \cup [\text{one} : \text{int list}] \vdash \text{let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let rec one} = 1 :: \text{one in let } x = 2 \text{ in fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow$
 int list

Proof for #1: Which rule?

?

$\Gamma \cup [\text{one} : \text{int list}] \vdash (1 :: \text{one}) : \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #1: This uses the application rule.

#3 #4

$\Gamma \cup [\text{one} : \text{int list}] \vdash (1 :: \text{one}) : \text{int list}$

#3 = $\Gamma \cup [\text{one} : \text{int list}] \vdash ((::)1) : \text{int list} \rightarrow \text{int list}$

#4 = $\Gamma \cup [\text{one} : \text{int list}] \vdash \text{one} : \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #3: What rule applies here?

?

$\Gamma \cup [\text{one} : \text{int list}] \vdash ((::)1) : \text{int list} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #3: This uses application again.

?

$(::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list} \quad 1 : \text{int}$

$\Gamma \cup [\text{one} : \text{int list}] \vdash ((::)1) : \text{int list} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #3: Since $(::)$ is predefined, we can treat this as a constant. 1 is also a constant. Thus, this part is done.

$(::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list} \quad 1 : \text{int}$

$\Gamma \cup [\text{one} : \text{int list}] \vdash ((::)1) : \text{int list} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #4: What rule applies here?

?

$\Gamma \cup [\text{one} : \text{int list}] \vdash \text{one} : \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #4: This is the variable rule.

$\Gamma \cup [\text{one} : \text{int list}] \vdash \text{one} : \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #2: What rule applies here?

$\Gamma \cup [\text{one} : \text{int list}] \vdash \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #2: We will use the let rule.

$\Gamma \cup [\text{one} : \text{int list}] \vdash 2 : \text{int}$ #5
 $\Gamma \cup [\text{one} : \text{int list}] \vdash \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$
 #5 = $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #5: What rule should we use?

$\Gamma \cup [\text{one} : \text{int list}, x : \text{int}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #5: Function rule

$\Gamma \cup [\text{one} : \text{int list}, x : \text{int}, y : \text{int}] \vdash (x :: y :: \text{one}) : \text{int list}$
 $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #5: Now what? Two more applications...

#6 #7
 $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}, y : \text{int}] \vdash (x :: y :: \text{one}) : \text{int list}$
 $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$
 #6 = $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}, y : \text{int}] \vdash ((::)x) : \text{int list} \rightarrow \text{int list}$
 #7 = $\Gamma \cup [\text{one} : \text{int list}, x : \text{int}, y : \text{int}] \vdash (y :: \text{one}) : \text{int list}$

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof for #6:

$\Gamma \cup [\text{one} : \text{int list}, x : \text{int}, y : \text{int}] \vdash ((::)x) : \text{int list} \rightarrow \text{int list}$

Types as Specifications

$\Gamma \vdash (\text{let } \text{rec } \text{one} = 1 :: \text{one} \text{ in } \text{let } x = 2 \text{ in } \text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int } \text{list}$

Proof for #7:

?

$\Gamma \cup [\text{one} : \text{int } \text{list}, x : \text{int}, y : \text{int}] \vdash (y :: \text{one}) : \text{int } \text{list}$

Type systems are a major area of research, and can be very involved, more so than we have shown here. Types can be seen as *specifications* of desired program behavior.

- ▶ Types describe properties
- ▶ Different type systems describe different properties, eg
 - ▶ Data is read-write versus read-only
 - ▶ Operation has authority to access data
 - ▶ Data came from “right” source
 - ▶ Operation might or could not raise an exception
- ▶ Common type systems focus on types describing data layout and access methods