

CS421 Topic 8: Type Checking¹

Sameer Sundresh
sundresh@uiuc.edu

University of Illinois at Urbana-Champaign

June 7, 2007

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

Objectives

- ▶ create a user-defined type describing the syntax of a language
- ▶ create a user-defined type describing the types of program expressions
- ▶ create a recursive function that acts as a type checker

Writing a type checker

Goal: write a function

- ▶ `type_of : type_env expr -> type`

To do that, we need:

- ▶ A user-defined type `expr`
- ▶ Another user-defined type `type`
- ▶ `type var_name = string`
- ▶ `type type_env = (var_name * type) list`
- ▶ The typing rules from our language (we'll translate them to code)

Note that typing rules for `fun` and `let rec` need to know the types of bound variables—so let's make the language explicitly typed.

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

Assumptions $\dots\dots\dots$

$$\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau$$

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

Conclusion $\dots\dots\dots$

- ▶ **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 Judgments Typing Rules Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

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 Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

Axioms – Variables

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:

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

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 Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

Simple Rules

- ▶ 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 Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

Simple Rules

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

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

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

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

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

Objectives Type Judgments Typing Rules Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism 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

Objectives Type Judgments Typing Rules Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

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:

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

Objectives Type Judgments Typing Rules Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

Function Application

Simple Rules

Objectives Type Judgments Typing Rules Polymorphism Conclusion	General Format Axioms Simple Rules Polymorphism Let and Letrec
----------------------------------------------------------------------------	----------------------------------------------------------------------------

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:

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

Objectives Type Judgments Typing Rules Polymorphism	Polymorphism Type Schemas Polymorphic Functions Polymorphic Function Application	Objectives Type Judgments Typing Rules Polymorphism	Polymorphism Type Schemas Polymorphic Functions Polymorphic Functions and Application Polymorphic Let and Letrec	Objectives Type Judgments Typing Rules Polymorphic Functions Conclusion
Type Instantiations		Polymorphic Let and Letrec		Conclusion

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:

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

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:

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

Hopefully you now have a better understanding of how type checking works.
 Next class, we will discuss how to automatically infer types, rather than requiring explicit type annotations on fun and let rec expressions.