

Lecture 10: Type Checking II

Yu Feng
Winter 2025

Outline

- We will talk about types in λ^+

Motivation

- When writing programs, everything is great as long as the program works.
- Unfortunately, this is usually not the case
- Programs crash, don't compute what we want them to compute, etc.
- This is arguably the **biggest problem** software faces today

Types

- In this class, we will focus on one kind of abstraction: types
- This means abstract values are the types in the language
- What is a type? An abstract value representing an (usually) infinite set of concrete values
- Question: For proving what kind of properties are types as abstract values useful?
- Answer: To avoid run-time type errors!

Adding types to λ^+

- Adding types to λ^+

$$T ::= T \rightarrow T \mid \text{Int} \mid \text{List}[T]$$

- Adding type annotation to λ^+ expressions:

$e ::=$...	(as before)
	$\text{Nil}[T]$	empty list
	$\text{lambda } x : T. e$	lambda abstraction
	$\text{fix } f : T \text{ is } e$	fixed-point operator
	$(e @ T)$	type annotation

Typing rules for integers

$$\frac{}{\Gamma \vdash i : \text{Int}} \text{T-INT}$$

Any integer constant i is of type **integer**

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \quad \square \in \{+, -, *\}}{\Gamma \vdash e_1 \square e_2 : \text{Int}} \text{T-ARITH}$$

if e_1 and e_2 are both integers, then
 $e_1 \square e_2$ will also be integer

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \quad \square \in \{<, >, =\}}{\Gamma \vdash e_1 \square e_2 : \text{Bool}} \text{T-REL}$$

if e_1 and e_2 are both integers, then
 $e_1 \square e_2$ will be a boolean

Typing rules for booleans

$$\frac{}{\Gamma \vdash \text{true} : \text{Bool}} \text{T-TRUE}$$

$$\frac{}{\Gamma \vdash \text{false} : \text{Bool}} \text{T-FALSE}$$

Boolean constants have type **Bool**

$$\frac{\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : T_1 \quad \Gamma \vdash e_3 : T_2 \quad T_1 = T_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T_1} \text{T-IF}$$

if e_1 is of type **Bool**, both e_2 and e_3 have the same type $T_1 = T_2$, then the whole if-else expression is of type T_1

Typing rules for lambda

$$\frac{x : T_1, \Gamma \vdash e : T_2}{\Gamma \vdash (\text{lambda } x : T_1. e) : T_1 \rightarrow T_2} \text{ T-LAMBDA}$$

If x is of type T_1 and e is of type T_2 ,
then the lambda expression has type $T_1 \rightarrow T_2$

$$\frac{\Gamma \vdash e_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash e_2 : T_3 \quad T_1 = T_3}{\Gamma \vdash (e_1 \ e_2) : T_2} \text{ T-APP}$$

if e_1 has type $T_1 \rightarrow T_2$, and e_2 has type T_3 such that $T_1 = T_3$
then lambda app return a value of type T_2

Typing rules for let-binding

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \text{ T-VAR}$$

If the type of x is T in the current type environment, then x is of type T

$$\frac{\Gamma \vdash e_1 : T_1 \quad x : T_1, \Gamma \vdash e_2 : T_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : T_2} \text{ T-LET}$$

if x and e_1 are of type T_1 , and under the extended environment, e_2 is of type T_2 , then the whole expression has type T_2

Typing rules for list

$$\frac{}{\Gamma \vdash \text{Nil}[T] : \text{List}[T]} \text{T-NIL}$$

An empty list is a list

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : \text{List}[T]}{\Gamma \vdash e_1 :: e_2 : \text{List}[T]} \text{T-CONS}$$

if e_1 is of type T and e_2 is of type $\text{List}[T]$,
then $e_1 :: e_2$ is of type $\text{List}[T]$

$$\frac{\Gamma \vdash e_1 : \text{List}[T_1] \quad \Gamma \vdash e_2 : T_2 \quad x : T_1, y : \text{List}[T_1], \Gamma \vdash e_3 : T_3 \quad T_2 = T_3}{\Gamma \vdash \text{match } e_1 \text{ with Nil} \rightarrow e_2 \mid x :: y \rightarrow e_3 \text{ end} : T_2} \text{T-MATCH}$$

If e is a list, and both the Nil branch and the cons branch have the same type,
then the overall pattern-match has that type.

Typing checking by example

$$\begin{array}{c}
 x:\text{int} \quad x+2:\text{int} \\
 \hline
 \lambda x:\text{int}. x+2 : \text{int} \rightarrow \text{int}
 \end{array}
 \quad
 \begin{array}{c}
 \text{wrong!} \\
 \text{y:List[int]} \\
 \text{y:List[int]}
 \end{array}
 \quad
 \text{List[int] = int}$$

$$\begin{array}{c}
 \text{y:List[int]} \vdash \\
 [1,2] : \text{List[int]}
 \end{array}
 \quad
 \begin{array}{c}
 \lambda x:\text{int}. x+2 \quad \text{y}
 \end{array}$$

$$\text{let } \underbrace{y = [1,2]}_{e_1} \text{ in } \underbrace{\lambda x:\text{int}. x+2 \quad \text{y}}_{e_2}$$

TODOs by next lecture

- TBD