Lecture 3: An Introduction to Types

Ankush Das

January 21, 2025

1 Introduction

Today, we will study how types can prevent programmers from making mistakes while writing programs. A type checker is simply a validity checker for a program to make sure we do not accidentally execute invalid programs.

2 The LL1 Type Checker

Let's return to the simple arithmetic and boolean expression language introduced in Lecture 1 and Homework 1. The syntax is written as

$$\text{Expressions} \quad e ::= e + e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \overline{n} \mid \mathsf{true} \mid \mathsf{false}$$

Since we have only two kinds of expressions in our language, we need two types

$$\text{Types} \quad \tau ::= \mathsf{int} \mid \mathsf{bool}$$

Now, we define the rules of the type system. We use a simple judgment written as $e : \tau$ meaning that expression e has type τ. Again, the principle for defining the type system is simple. We take every expression defined in the syntax and we define the rule for typing that expression. The rules are defined as

$$\frac{e_1 : \mathsf{int} \qquad e_2 : \mathsf{int}}{e_1 + e_2 : \mathsf{int}}\ \text{T-Add} \qquad \frac{e : \mathsf{bool} \qquad e_1 : \tau \qquad e_2 : \tau}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}\ \text{T-If} \qquad \frac{}{\overline{n} : \mathsf{int}}\ \text{T-Num}$$

$$\frac{}{\mathsf{true} : \mathsf{bool}}\ \text{T-TT} \qquad \frac{}{\mathsf{false} : \mathsf{bool}}\ \text{T-FF}$$

The T-Add rule defines that $e_1 + e_2$ can only have type int and for that e_1 and e_2 must have type int. The T-If rule dictates that e must have type bool and e_1 and e_2 must have the same arbitrary type τ and then the 'if' expression has the same type τ. The remaining rules just describe how to type values.

An interesting thing to note in this language is that there are no variables. That's because there is no way to introduce variables. Now, we can either introduce functions (like λ-calculus) or we can introduce a let expression: $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$, meaning x will have value that e_1 evaluates to in the expression e_2.

How do we type this expression?

$$\frac{e_1 : \tau' \qquad e_2 : \tau?}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}\ \text{T-Let}$$

This doesn't seem entirely correct. The expression e_2 can refer to variable x but that is not specified as a premise when typing e_2.

Thus, we need to introduce a typing context, which we call Γ that tracks the types of all the variables in the context (e.g., $\Gamma = \{x_1 : \tau_1, x_2 : \tau_2, \ldots\}$). We mandate that all variables in Γ are

distinct. We slightly modify the typing judgment as $\Gamma \vdash e : \tau$ meaning that expression e has type τ in the presence of context Γ, i.e., in the presence of $x_1 : \tau_1, x_2 : \tau_2, \ldots$. Thus, we re-write the T-Let rule

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}\ \textsc{T-Let}$$

Now that we have studied types, we can prove a preservation theorem that states that a well-typed expression steps to another well-typed expression with the same type.

Theorem 1 (Preservation)**.** *If $\Gamma \vdash e : \tau$ and $e \mapsto e'$, then $\Gamma \vdash e' : \tau$.*

Proof. We will cover only a couple of cases of the proof. The proof proceeds by rule induction on the semantics judgment: $e \mapsto e'$.

- Case when $e = e_1 + e_2$ and

  $$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2}\ \textsc{Add-L}$$

  From the typing of e, we get

  $$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \textsc{Add}$$

  Applying inversion on this rule, we get $\Gamma \vdash e_1 : \mathsf{int}$ and $e_1 \mapsto e_1'$. Appealing to the inductive hypothesis, we obtain $\Gamma \vdash e_1' : \mathsf{int}$. Now, applying the Add typing rule, we get

  $$\frac{\Gamma \vdash e_1' : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1' + e_2 : \mathsf{int}}\ \textsc{Add}$$

  Hence $\Gamma \vdash e' : \mathsf{int}$ when $\Gamma \vdash e : \mathsf{int}$, proving preservation.

- Case when $e = e_1 + e_2$ and

  $$\frac{e_1\ \mathsf{value} \qquad e_2 \mapsto e_2'}{e_1 + e_2 \mapsto e_1' + e_2}\ \textsc{Add-R}$$

  From the typing of e, we get

  $$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \textsc{Add}$$

  Applying inversion on these rules, we get $e_2 \mapsto e_2'$ and $\Gamma \vdash e_2 : \mathsf{int}$. Appealing to the inductive hypothesis, we obtain $\Gamma \vdash e_2' : \mathsf{int}$. Now, applying the Add typing rule, we get

  $$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2' : \mathsf{int}}{\Gamma \vdash e_1 + e_2' : \mathsf{int}}\ \textsc{Add}$$

  Hence $\Gamma \vdash e' : \mathsf{int}$ when $\Gamma \vdash e : \mathsf{int}$, proving preservation.

- Case when $e = \overline{n_1} + \overline{n_2}$ and

  $$\frac{}{\overline{n_1} + \overline{n_2} \mapsto \overline{n_1 \oplus n_2}}\ \textsc{Add-V}$$

  From the typing of e, we get

  $$\frac{\Gamma \vdash \overline{n_1} : \mathsf{int} \qquad \Gamma \vdash \overline{n_2} : \mathsf{int}}{\Gamma \vdash \overline{n_1} + \overline{n_2} : \mathsf{int}}\ \textsc{Add}$$

  This is a base case, there's no need inductive hypothesis to apply here. But we can simply use Add rule to get

  $$\frac{}{\Gamma \vdash \overline{n_1 \oplus n_2} : \mathsf{int}}\ \textsc{Add}$$

  because all numbers are values and have type int. Hence $\Gamma \vdash e' : \mathsf{int}$ when $\Gamma \vdash e : \mathsf{int}$, proving preservation.

The readers are encouraged to carry out the remaining cases themselves. □