Objectives

- Explain the parts of a type judgment.
- Build proof trees to indicate the derivation of a type for a program.
- Explain the circumstances under which a type environment can be modified.

The Language

- We are going to type λ-calculus extended with let, if, arithmetic, and comparisons.

Format of a Type Judgment

A *type judgment* has the following form:

\[ \Gamma \vdash e : \alpha \]

where \( \Gamma \) is a *type environment*, \( e \) is some expression, and \( \alpha \) is a *type*.

- \( \Gamma \vdash \text{if true then 4 else 38 : Int} \)
- \( \Gamma \vdash \text{true && false : Bool} \)

Note: the \( \vdash \) is pronounced “turnstile” or “entails”.

The Language

\[
\begin{align*}
L &::= \lambda x.L \\
& \mid L.L \\
& \mid \text{let } x = L \text{ in } L \\
& \mid \text{if } L \text{ then } L \text{ else } L \\
E &::= x \\
& \mid n \\
& \mid b \\
& \mid E \oplus E \\
& \mid E \sim E \\
& \mid E \&\& E \\
& \mid E \| E
\end{align*}
\]

- abstractions
- applications
- Let expressions
- If expressions
- expressions
- variables
- integers
- booleans
- integer operations
- integer comparisons
- boolean and
- boolean or
The Parts of a Rule

Assumptions on top

\[ \Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \]

\[ \Gamma \vdash e_1 + e_2 : \text{Int} \]

Conclusion on the bottom

- If a rule has no assumptions, then it is called an axiom.
- \( \Gamma \) is a set of the form \( \{ x : \alpha ; \ldots \} \).
- \( \Gamma \) may be left out if we don’t need a type environment.

**Basic Idea:** The meaning of an expression can be determined by combining the meaning of its parts.

### Axioms

**Constants**

\[ \vdash n : \text{Int} \], when \( n \) is an integer.

\[ \vdash \text{true} : \text{Bool} \]

\[ \vdash \text{false} : \text{Bool} \]

**Variables**

\[ \Gamma \vdash x : \alpha, \text{if} x : \alpha \in \Gamma \]

- Here, \( \alpha \) is a type variable; it stands for another type.
- These are rules that are true no matter what the context is.

### Simple Rules

**Arithmetic**

\[ \Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \]

\[ \Gamma \vdash e_1 + e_2 : \text{Int} \]

**Relations**

\[ \Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \]

\[ \Gamma \vdash e_1 \sim e_2 : \text{Bool} \]

**Booleans**

\[ \Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : \text{Bool} \]

\[ \Gamma \vdash e_1 \&\& e_2 : \text{Bool} \]

\[ \Gamma \vdash e_1 \mid\!\mid e_2 : \text{Bool} \]

### Example 0

Suppose we want to prove that \( \Gamma \vdash (x \ast 5 > 7) \&\& y : \text{Bool} \).

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.

\[ \Gamma \vdash (x \ast 5 > 7) \&\& y : \text{Bool} \]

Look at the outermost expression. What rule applies here?
Example 0

Suppose we want to prove that $\Gamma \vdash (x \ast 5 > 7) \& \& y : \text{Bool}$.
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.

$$\Gamma \vdash (x \ast 5 > 7) \& \& y : \text{Bool}$$

Look at the outermost expression. What rule applies here?

$$\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : \text{Bool}$$

$$\Gamma \vdash e_1 \& \& e_2 : \text{Bool}$$

What to do next? Let’s work left to right. The expression we want next is a “greater” expression. (Besides, the $y$ expression is already an axiom.)

Following the “greater” rule, we break the $x \ast 5 > 7$ into two parts.

$$\Gamma \vdash x \ast 5 : \text{Int} \quad \Gamma \vdash 7 : \text{Int}$$

$$\Gamma \vdash x \ast 5 > 7 : \text{Bool} \quad \Gamma \vdash y : \text{Bool}$$

$$\Gamma \vdash (x \ast 5 > 7) \& \& y : \text{Bool}$$

We will turn our attention to the multiplication now.

At this point, there are no more subtrees to expand out. We are done.

$$\Gamma \vdash x : \text{Int} \quad \Gamma \vdash 5 : \text{Int}$$

$$\Gamma \vdash x \ast 5 : \text{Int} \quad \Gamma \vdash 7 : \text{Int}$$

$$\Gamma \vdash x \ast 5 > 7 : \text{Bool} \quad \Gamma \vdash y : \text{Bool}$$

$$\Gamma \vdash (x \ast 5 > 7) \& \& y : \text{Bool}$$
Type Variables in Rules

A monotype $\tau$ can be a

- type constant (e.g., Int, Bool, etc.)
- instantiated type constructor (e.g., [Int], Int $\rightarrow$ Int)
- a type variable $\alpha$

If Rule

\[
\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : \alpha \quad \Gamma \vdash e_3 : \alpha
\]
\[
\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \alpha
\]

- Here, $\alpha$ is a meta-variable.
- This 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.

Function Application

\[
\Gamma \vdash e_1 : \alpha_2 \rightarrow \alpha \quad \Gamma \vdash e_2 : \alpha_2
\]
\[
\Gamma \vdash e_1 \ e_2 : \alpha
\]

- If you have a function of type $\alpha_2 \rightarrow \alpha$ and an argument $e_2$ of type $\alpha_2$, then applying $e_1$ to $e_2$ will produce an expression of type $\alpha$.
- You can generalize this rule to multiple arguments.

\[
\Gamma \vdash \text{incList : [Int] \rightarrow [Int]} \quad \Gamma \vdash xx : \text{[Int]}
\]
\[
\Gamma \vdash \text{incList } xx : \text{[Int]}
\]

Function Rule

\[
\Gamma \cup \{x : \alpha_1\} \vdash e : \alpha_2
\]
\[
\Gamma \vdash \lambda x. e : \alpha_1 \rightarrow \alpha_2
\]

- Important point: this rule describes types, and also describes when you may change $\Gamma$.
- You may NOT change $\Gamma$ except as described!

Example: show that $\{\} \vdash \lambda x. x + 1 : \text{Int} \rightarrow \text{Int}$.

\[
\{\} \vdash \lambda x. x + 1 : \text{Int} \rightarrow \text{Int}
\]
## Function Rule

\[
\Gamma \cup \{x : \alpha_1\} \vdash e : \alpha_2 \\
\Gamma \vdash \lambda x. e : \alpha_1 \rightarrow \alpha_2
\]

- Important point: this rule describes types, and also describes when you may change \(\Gamma\).
- You may **NOT** change \(\Gamma\) except as described!

\[
\{x : \text{Int}\} \vdash x + 1 : \text{Int} \\
\{\} \vdash \lambda x. x + 1 \rightarrow \text{Int}
\]

## Let Rule

- Here is `let`. Note that **Haskell** uses the recursive rule, and it is polymorphic.

\[
\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \cup \{x : \tau_1\} \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}
\]

\[
\frac{\Gamma \cup \{x : \tau_1\} \vdash e_1 : \tau_1 \quad \Gamma \cup \{x : \tau_1\} \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}
\]

## Example 1 — Proof

Prove that \(\Gamma \vdash (\lambda x. \lambda y. x + y) : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}\).
Assume that \(\Gamma = \{\}\).
Prove that $\Gamma \vdash (\lambda x.\lambda y.x + y) : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$.
Assume that $\Gamma = \{\}$

{} $\vdash (\lambda x.\lambda y.x + y) : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$
Example 1 — inferencing

Infer the type of \((\lambda x.\lambda y.x + y)\).
Assume that \(\Gamma = \{\}\)

\[
\begin{align*}
\{x : \beta\} \vdash (\lambda y.x + y) : \gamma & \\
\{\} \vdash (\lambda x.\lambda y.x + y) : \alpha \equiv \beta \rightarrow \gamma
\end{align*}
\]
Example 1 — inferencing

Infer the type of \( (\lambda x. \lambda y. x + y) \).
Assume that \( \Gamma = {} \)

\[
\begin{align*}
\Gamma' \vdash x : \text{Int} & \quad \Gamma' \vdash y : \text{Int} \\
\Gamma' \equiv \{x : \text{Int}, y : \text{Int}\} & \vdash x + y : \text{Int} \\
\{x : \text{Int}\} & \vdash (\lambda y. x + y) : \text{Int} \rightarrow \text{Int} \\
\{\} & \vdash (\lambda x. \lambda y. x + y) : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}
\end{align*}
\]

Example 2

\[\Gamma \vdash \text{let id} = \lambda x. x \text{ in id (id 10)} : \text{Int}\]

\[\Gamma \vdash \lambda x. x : \text{Int} \rightarrow \text{Int} \quad \Gamma' \vdash \text{id (id 10)} : \text{Int}\]

\[\Gamma \vdash \text{let id} = \lambda x. x \text{ in id (id 10)} : \text{Int}\]

Let \( \Gamma' \equiv \Gamma \cup \{\text{id : Int} \rightarrow \text{Int}\} \)
Example 2

\[
\begin{align*}
\Gamma \cup \{x : \text{Int}\} &\vdash x : \text{Int} \\
\Gamma &\vdash \lambda x. x : \text{Int} \to \text{Int} \\
\Gamma' &\vdash \text{id} : \text{Int} \to \text{Int} \\
\Gamma' &\vdash \text{id} \; \text{id} \; 10 : \text{Int} \\
\Gamma &\vdash \text{let id} = \lambda x. x \; \text{in id (id 10)} : \text{Int}
\end{align*}
\]

Let \(\Gamma' \equiv \Gamma \cup \{\text{id} : \text{Int} \to \text{Int}\}\)