



Compilers

Type Environments

 $\vdash \text{false} : \text{Bool}$

[False]

s is a string literal

 $\vdash s : \text{String}$

[String]

`new T` produces an object of type `T`

– Ignore `SELF_TYPE` for now . . .

$$\frac{}{\vdash \text{new } T : T} \quad [\text{New}]$$

$$\frac{\vdash e: \text{Bool}}{\vdash !e : \text{Bool}} \quad [\text{Not}]$$

$$\frac{\begin{array}{c} \vdash e_1: \text{Bool} \\ \vdash e_2:T \end{array}}{\vdash \text{while } e_1 \text{ loop } e_2 \text{ pool:Object}} \quad [\text{Loop}]$$

- What is the type of a variable reference?

$$\frac{x \text{ is a variable}}{\vdash x: ?} \quad [\text{Var}]$$

- The local, structural rule does not carry enough information to give x a type.

- Put more information in the rules!
- *A type environment gives types for free variables*
 - A type environment is a function from **ObjectIdentifiers** to **Types**
 - A variable is free in an expression if it is not defined within the expression

Let O be a function from **ObjectIdentifiers** to **Types**

The sentence $O \vdash e : T$

is read: Under the assumption that variables have the types given by O , it is provable that the expression e has the type T

The type environment is added to the earlier rules:

$$\frac{\text{i is an integer literal}}{O \vdash i : \text{Int}} \quad [\text{Int}]$$

$$\frac{O \vdash e_1 : \text{Int} \quad O \vdash e_2 : \text{Int}}{O \vdash e_1 + e_2 : \text{Int}} \quad [\text{Add}]$$

And we can write new rules:

$$\frac{O(x) = T}{\vdash x : T} \quad [\text{Var}]$$

$$\frac{O[T_0/x] \vdash e_1 : T_1}{O \vdash \text{let } x:T_0 \text{ in } e_1 : T_1} \quad [\text{Let-No-Init}]$$

Fill in the correct type environments in the following type rule

$$\frac{O_1 \vdash e_1 : T_1 \quad O_2 \vdash e_2 : T_2}{O \vdash \text{let } x : T_1 \leftarrow e_1 \text{ in } e_2 : T_2}$$

[Let-Init]

- ☐ $O_1 = O[T_1/x]; O_2 = O[T_1/x]$
- ☐ $O_1 = O[T_1/x]; O_2 = O[T_2/x]$
- ☐ $O_1 = O; O_2 = O[T_1/x]$
- ☐ $O_1 = O; O_2 = O[T_2/x]$

- The type environment gives types to the free identifiers in the current scope
- The type environment is passed down the AST from the root towards the leaves
- Types are computed up the AST from the leaves towards the root