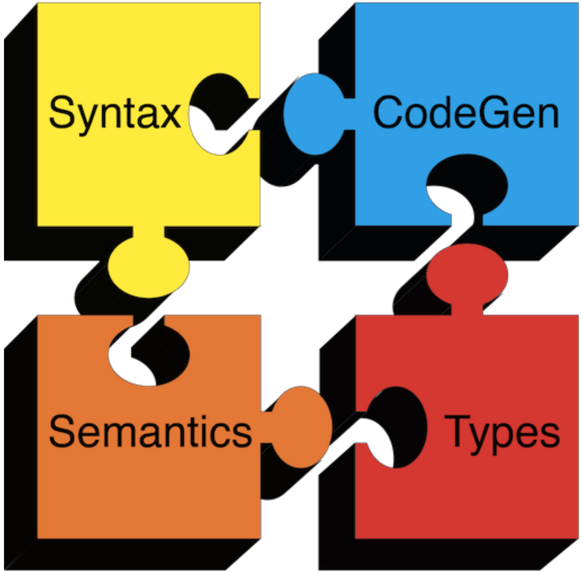


# Compilers

## Error Recovery

Alex Aiken

# Error Recovery

- As with parsing, it is important to recover from type errors

- Detecting where errors occur is easier than in parsing
  - There is no reason to skip over portions of code

- The problem:
  - What type is assigned to an expression with no legitimate type?
  - This type will influence the typing of the enclosing expression

# Error Recovery

- Assign type Object to ill-typed expressions

$$\text{let } y : \text{Int} \leftarrow \underset{x:\text{Object}}{\overset{\text{Object}}{x + 2}} \text{ in } y + 3$$

error: x is undefined

error: + applied to Object

error: bad assignment

⇒ a workable solution but with cascading errors

# Error Recovery

- Introduce a new type No_type for use with ill-typed expressions

$c \le \text{Object}$

- Define No_type $\le$ C for all types C

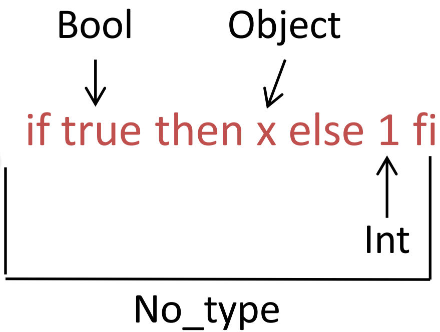
- Every operation is defined for No_type
  - With a No_type result

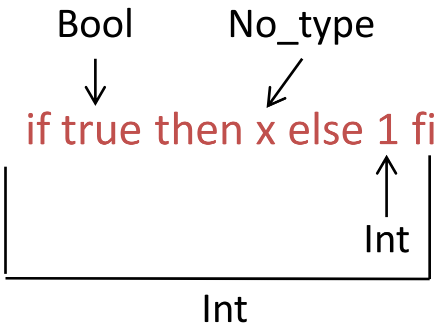
$$\text{let } y : \text{Int} \leftarrow \overbrace{\underset{x:\text{No\_type}}{\underline{x}} + 2}^{\text{No\_type}} \text{ in } y + 3$$

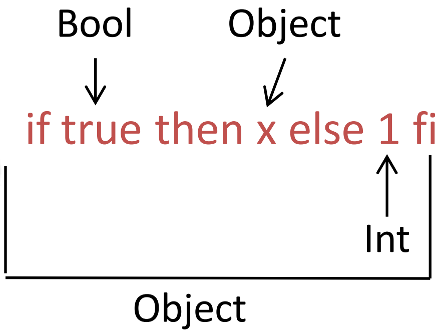
error: x is undefined

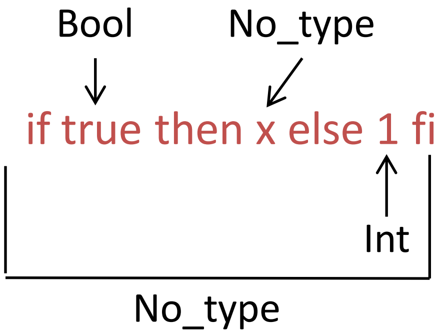
# Error Recovery

Choose the correct labeling of types for the code fragment, using No_type as described in the video. Assume that x is not defined.

○ 

○ 

○ 

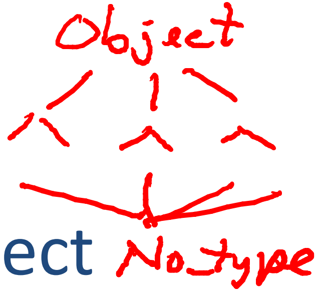
○ 

# Error Recovery

- A “real” compiler would use something like No_type

- However, there are some implementation issues
  - The class hierarchy is not a tree anymore

No_type ≤ C



- The Object solution is fine in the course project