



# Compilers

---

## Type Checking

- We have seen two examples of formal notation specifying parts of a compiler
  - Regular expressions
  - Context-free grammars
- The appropriate formalism for type checking is logical rules of inference

- Inference rules have the form  
*If Hypothesis is true, then Conclusion is true*
- Type checking computes via reasoning  
*If  $E_1$  and  $E_2$  have certain types, then  $E_3$  has a certain type*
- Rules of inference are a compact notation for “If-Then” statements

- The notation is easy to read with practice
- Start with a simplified system and gradually add features
- Building blocks
  - Symbol  $\wedge$  is “and”
  - Symbol  $\Rightarrow$  is “if-then”
  - $x:T$  is “ $x$  has type  $T$ ”

If  $e_1$  has type  $\text{Int}$  and  $e_2$  has type  $\text{Int}$ , then  $e_1 + e_2$   
has type  $\text{Int}$

$(e_1 \text{ has type } \text{Int} \wedge e_2 \text{ has type } \text{Int}) \Rightarrow e_1 + e_2 \text{ has type } \text{Int}$

$(e_1 : \text{Int} \wedge e_2 : \text{Int}) \Rightarrow e_1 + e_2 : \text{Int}$

The statement

$$(e_1: \text{Int} \wedge e_2: \text{Int}) \Rightarrow e_1 + e_2: \text{Int}$$

is a special case of

$$\text{Hypothesis}_1 \wedge \dots \wedge \text{Hypothesis}_n \Rightarrow \text{Conclusion}$$

This is an inference rule

- By tradition inference rules are written

$$\frac{\vdash \text{Hypothesis} \dots \vdash \text{Hypothesis}}{\vdash \text{Conclusion}}$$

- Cool type rules have hypotheses and conclusions

$$\vdash e:T$$

- $\vdash$  means “it is provable that . . .”

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

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



- These rules give templates describing how to type integers and + expressions
- By filling in the templates, we can produce complete typings for expressions

$$\frac{\frac{1 \text{ is an int literal}}{\vdash 1 : \text{Int}} \quad \frac{2 \text{ is an int literal}}{\vdash 2 : \text{Int}}}{\vdash 1 + 2 : \text{Int}}$$

- A type system is *sound* if
  - Whenever  $\vdash e : T$
  - Then  $e$  evaluates to a value of type  $T$
- We only want sound rules
  - But some sound rules are better than others!

Choose the type rules that are sound

## Type Checking

 $\vdash e_1 : T_1$  $\vdash e_2 : T_2$  $\vdots$  $\vdash e_n : T_n$ 

[Sequence]

 $\vdash \{e_1; e_2; \dots e_n\} : T_n$  $\vdash e_1 : \text{Int} \quad \vdash e_2 : \text{Int}$  $\vdash e_1 / e_2 : \text{Bool}$ 

[Divide]

 $\vdash e_1 : \text{Int} \quad \vdash e_2 : \text{Int}$  $\vdash e_1 < e_2 : \text{Int}$ 

[Compare]

 $\vdash e_1 : T_1$  $\vdash \text{isvoid } e_1 : \text{Bool}$ 

[Isvoid]

- Type checking proves facts  $e: T$ 
  - Proof is on the structure of the AST
  - Proof has the shape of the AST
  - One type rule is used for each AST node
- In the type rule used for a node  $e$ :
  - Hypotheses are the proofs of types of  $e$ 's subexpressions
  - Conclusion is the type of  $e$
- Types are computed in a bottom-up pass over the AST