

# Compilers

- 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 ∧ is "and"
  - Symbol ⇒ is "if-then"
  - x:T is "x has type T"

then 
$$e_1 + e_2$$

(
$$e_1$$
 has type Int  $\land e_2$  has type Int)  $\Rightarrow$  type Int

$$e_1 + e_2$$
 has

$$(e_1: Int \land e_2: Int) \Rightarrow e_1 + e_2: Int$$

#### The statement

$$(e_1: Int \land e_2: Int) \Rightarrow e_1 + e_2: Int$$

is a special case of

 $\mathsf{Hypothesis}_1 \land \ldots \land \mathsf{Hypothesis}_n \Rightarrow \mathsf{Conclusion}$ 

This is an inference rule

By tradition inference rules are written

⊢ Conclusion

Cool type rules have hypotheses and conclusions

means "it is provable that . . ."

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

 These rules give templates describing how to type integers and + expressions

 By filling in the templates, we can produce complete typings for expressions

1 is an int literal 2 is an int literal 
$$\vdash$$
 1 : Int  $\vdash$  2 : Int  $\vdash$  1 + 2 : Int

- A type system is sound if
  - Whenever ⊢ 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<sub>1</sub>: T<sub>1</sub>

$$\vdash e_1: I_1$$
  
 $\vdash e_2: T_2$ 

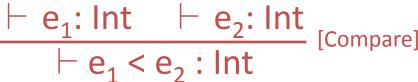
$$\vdash e_n: T_n$$
 $\vdash \{e_1; e_2; ... e_n; \}: T_n$ 

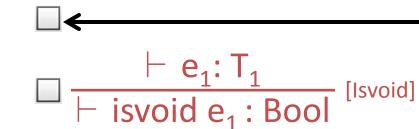


[Sequence]









- 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