



Compilers

Operational Semantics

- Once again we introduce a formal notation
- Logical rules of inference, as in type checking

- Recall the typing judgment


$$\underline{\text{Context}} \vdash \underline{e} : \underline{C}$$

*In the given **context**, expression **e** has type **C***

- We use something similar for evaluation

$$\underline{\text{Context}} \vdash \underline{e} : \underline{v}$$

*In the given **context**, expression **e** evaluates to value **v***


$$\frac{\text{Context} \vdash e_1 : 5 \quad \text{Context} \vdash e_2 : 7}{\text{Context} \vdash e_1 + e_2 : 12}$$

- Consider the evaluation of $y \leftarrow x + 1$
 - We track variables and their values with:
 - An environment : where in memory a variable is
 - A store : what is in the memory
- vars → memory locations
- memory locs → values

- A variable environment maps variables to locations
 - Keeps track of which variables are in scope
 - Tells us where those variables are

$$E = [\underline{a} : \underline{l_1}, \underline{b} : \underline{l_2}]$$

- A store maps memory locations to values

$$S = [\underline{l_1} \rightarrow \underline{5}, \underline{l_2} \rightarrow \underline{7}]$$

- $\underline{S'} = \underline{S}[\underline{12}/\underline{l_1}]$ defines a store S' such that

$$\underline{S'}(\underline{l_1}) = \underline{12} \quad \text{and} \quad \underline{S'}(\underline{l}) = \underline{S(l)} \text{ if } \underline{l} \neq \underline{l_1}$$

- Cool values are objects
 - All objects are instances of some class
- $X(a_1 = l_1, \dots, a_n = l_n)$ is a Cool object where
 - X is the class of the object
 - a_i are the attributes (including inherited ones)
 - l_i is the location where the value of a_i is stored

- Special cases (classes without attributes)

Int(5) the integer 5
Bool(true) the boolean true
String(4, "Cool") the string "Cool" of length 4

- There is a special value void of type **Object**
 - No operations can be performed on it
 - Except for the test isvoid
 - Concrete implementations might use NULL here

- The evaluation judgment is

$$\boxed{so, E, S \vdash e : v, S'}$$

- Given **so** the current value of **self**
- And **E** the current variable environment
- And **S** the current store
- If the evaluation of **e** terminates then
- The value of **e** is **v**
- And the new store is **S'**

- “Result” of evaluation is a value and a ^{new} store
 - New store models the side-effects
- Some things don't change
 - The variable environment
 - The value of self
 - The operational semantics allows for non-terminating evaluations