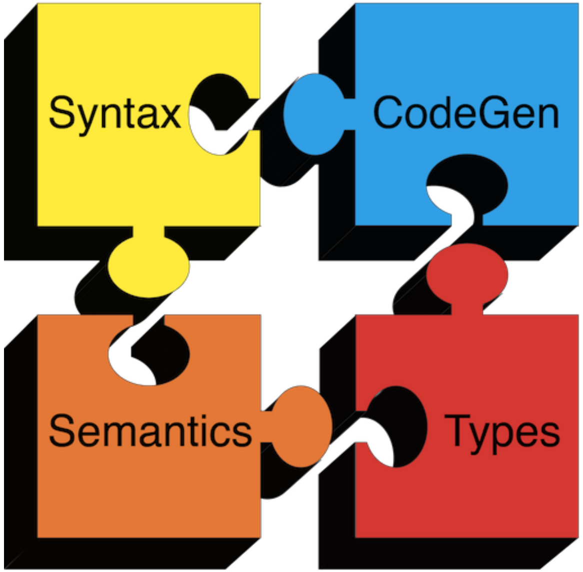


# Compilers

## Self Type Checking

Alex Aiken

# Self Type Checking

- SELF_TYPE's meaning depends on the enclosing class

$$O,M,C \vdash e : T$$

*An expression e occurring in the body of C has static type T given a variable type environment O and method signatures M*

# Self Type Checking

- The next step is to design type rules using SELF_TYPE

- Most of the rules remain the same
  - But use the new $\leq$ and lub

$$\frac{\begin{array}{l} O(Id) = T_0 \\ O,M,C \vdash e_1 : T_0 \\ T_1 \leq T_0 \end{array}}{O,M,C \vdash Id \leftarrow e_1 : T_1}$$

- Recall the old rule for dispatch

$$\frac{\begin{array}{l} O,M,C \vdash e_0 : T_0 \\ \quad\vdots \\ O,M,C \vdash e_n : T_n \\ M(T_0, f) = (T_1',\ldots,T_n',T_{n+1}') \\ T_{n+1}' \neq \text{SELF\_TYPE} \\ T_i \leq T_i' \qquad 1 \leq i \leq n \end{array}}{O,M,C \vdash e_0.f(e_1,\ldots,e_n) : T_{n+1}'}$$

# Self Type Checking

- If the return type of the method is SELF_TYPE then the type of the dispatch is the type of the dispatch expression:

$$
\frac{
\begin{array}{l}
O,M,C \vdash e_0 : T_0 \\
\qquad\quad \vdots \\
O,M,C \vdash e_n : T_n \\
M(T_0, f) = (T_1',\ldots,T_n', \text{SELF\_TYPE}) \\
T_i \leq T_i' \qquad\qquad 1 \leq i \leq n
\end{array}
}{
O,M,C \vdash e_0.f(e_1,\ldots,e_n) : T_0
}
$$

# Self Type Checking

- Formal parameters cannot be SELF_TYPE
- Actual arguments can be SELF_TYPE
  - The extended $\leq$ relation handles this case
- The type $T_0$ of the dispatch expression could be SELF_TYPE
  - Which class is used to find the declaration of f?
  - Answer: it is safe to use the class where the dispatch appears

# Self Type Checking

- Recall the original rule for static dispatch

$$\frac{\begin{array}{l} O,M,C \vdash e_0 : T_0 \\ \quad\vdots \\ O,M,C \vdash e_n : T_n \\ T_0 \leq T \\ M(T, f) = (T_1',\ldots,T_n',T_{n+1}') \\ T_{n+1}' \neq \text{SELF\_TYPE} \\ T_i \leq T_i' \qquad 1 \leq i \leq n \end{array}}{O,M,C \vdash e_0@T.f(e_1,\ldots,e_n) : T_{n+1}'}$$

# Self Type Checking

- If the return type of the method is SELF_TYPE we have:

$$\frac{\begin{array}{l} O,M,C \vdash e_0 : T_0 \\ \quad\quad\quad\quad \vdots \\ O,M,C \vdash e_n : T_n \\ T_0 \leq T \\ M(T, f) = (T_1',\ldots,T_n',SELF\_TYPE) \\ T_i \leq T_i' \quad\quad\quad 1 \leq i \leq n \end{array}}{O,M,C \vdash e_0@T.f(e_1,\ldots,e_n) : T_0}$$

- Why is this rule correct?

- If we dispatch a method returning SELF_TYPE in class T, don't we get back a T?

- No. SELF_TYPE is the type of the self parameter, which may be a subtype of the class in which the method appears

# Self Type Checking

- There are two new rules using SELF_TYPE

$$\frac{}{O,M,C \vdash \text{self} : \text{SELF\_TYPE}_C}$$

$$\frac{}{O,M,C \vdash \text{new SELF\_TYPE} : \text{SELF\_TYPE}_C}$$

# Self Type Checking

Choose the static/dynamic type pairs that are correct. For dynamic type, assume execution has halted at line 15.

| | Var | Static Type | Dynamic Type |
|---|---|---|---|
| ☐ | w | Animal | Pet |
| ☐ | x | Animal | Lion |
| ☐ | y | Pet | Pet |
| ☐ | z | Animal | Dog |

```
1  class Animal {
2      clone():SELF_TYPE {new SELF_TYPE}
3  }
4  class Pet inherits Animal {
5      clone():SELF_TYPE {new SELF_TYPE}
6  }
7  class Cat inherits Pet { … }
8  class Dog inherits Pet { … }
9  class Lion inherits Animal { … }
10 class Main {
11     w:Animal<-(new Animal).clone();
12     x:Animal<-(new Lion).clone();
13     y:Pet<-(new Cat).clone();
14     z:Animal<-(new Dog)@Animal.clone();
15     …
16 }
```

# Self Type Checking

- The extended $\leq$ and lub operations can do a lot of the work.

- SELF_TYPE can be used only in a few places. Be sure it isn't used anywhere else.

- A use of SELF_TYPE always refers to any subtype of the current class
  - The exception is the type checking of dispatch. The method return type of SELF_TYPE might have nothing to do with the current class

# Self Type Checking

- SELF_TYPE is a research idea
  - It adds more expressiveness to the type system
- SELF_TYPE is itself not so important
  - except for the project
- Rather, SELF_TYPE is meant to illustrate that type checking can be quite subtle
- In practice, there should be a balance between the complexity of the type system and its expressiveness