Any questions about scoping? Moving on...

1 Typing Rules, Judgements, and Checking

Some simple exercises for the class:

What is the type judgement for arithmetic (adding)?
What is the typing judgement for IF statements?
What does the following rule mean? How do we read it? Discuss!

\[
\begin{align*}
O, M, C & \vdash e_1 : T_1 \\
O, M, C & \vdash e_2 : T_2 \\
& \vdots \\
O, M, C & \vdash e_n : T_n \\
\hline
O, M, C & \vdash \{ e_1; e_2; \ldots e_n; \} : T_n
\end{align*}
\]

...That’s all very nice. But how on earth does that turn into code? PA4, after all, includes a ”type checker,” which really just enforces the rules we’ve written in Greek.

2 SELF_TYPE

What’s the difference between static and dynamic? As applied to type systems? As applied to dispatch?
In our type system, we define soundness as follows:

\[ \forall E. \text{ dynamic	extunderscore type}(E) \leq \text{ static	extunderscore type}(E) \]

Why does this work?

Consider the following piece of code:

Listing 1: SELF\_TYPE problem

```plaintext
class A {
    modifying\_op() : A {
        { (* ... *)
            self;
        }
    }
};

myObject : B <- (new B).modifying\_op(); (* badness *)
```

What’s wrong here? Solution: we add SELF\_TYPE to the language, and change our code:

Listing 2: SELF\_TYPE solution

```plaintext
modifying\_op() : SELF\_TYPE {
    { (* ... *)
        self;
    }
}
```

Note: Unlike other typenames, SELF\_TYPE means different things in different parts of the code. What changes do we need to make to our type system for that to work?

Which of the following are legal?

Listing 3: SELF\_TYPE placement

```plaintext
(* 1 *) class SELF\_TYPE inherits Q ...

(* 2 *) s : SELF\_TYPE <- ...

(* 3 *) s <- new SELF\_TYPE
```

Can SELF\_TYPE be a method argument type?
Listing 4: Using SELF_TYPE as an argument

class Main inherits IO {
    main() : Object {
        let p : A <- new A in
        p.test(p)
    };
};

class A { test(a : SELF_TYPE) : Int { 0 }; };

class B inherits A { test(a : SELF_TYPE) : Int { 1 }; };