CS 252:

Advanced Programming Language Principles

Typed
Arith

Prof. Tom Austin
San José State University

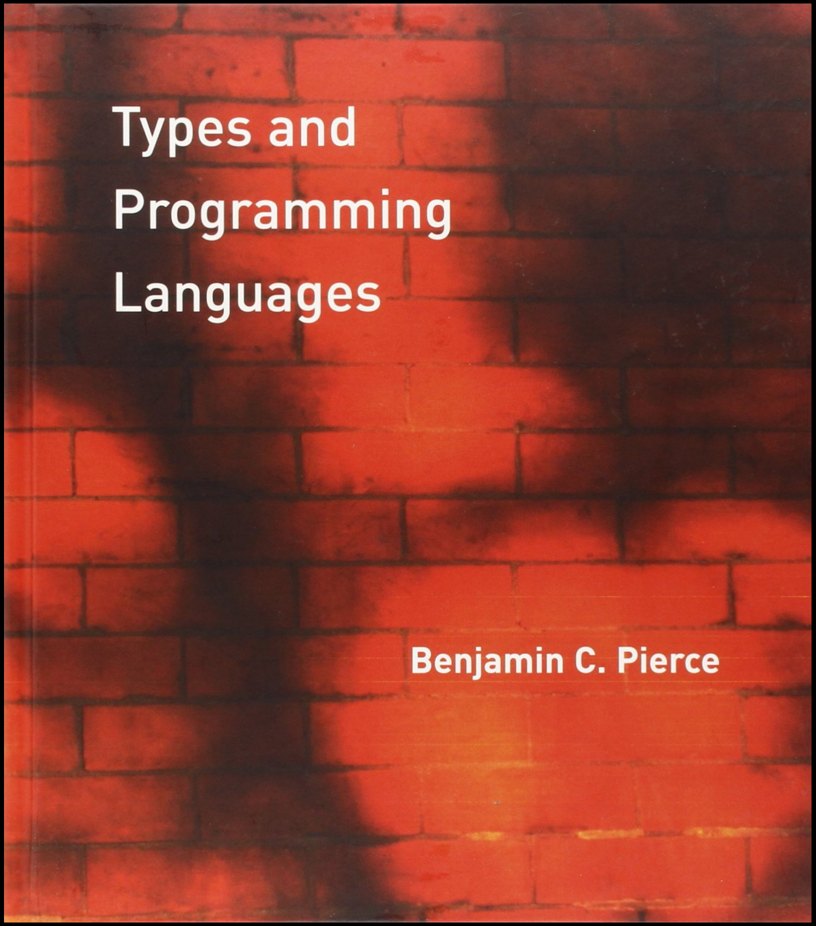
What do type systems give us?

- Tips for compilers to make code more efficient
- Tips for IDEs and other tools to make writing code easier
- Enforced documentation
- But most importantly…

Type systems prevent us from running code with errors.

Types & Programming Languages (TAPL)

- Standard reference
- Copies available at the library
- Chapter 8



Arith Language

```
e ::= true
    | false
    | 0
    | succ e
    | pred e
    | iszero e
    | if e then e
           else e
```

```
v ::= true
    | false
    | nv
nv ::= 0
     | succ nv
```

Small-step evaluation rules for Arith (in-class)

Types for Arith

Our typing rules will be of the form

$$\texttt{e:T}$$

This says that an expression `e` will either

1. evaluate to a value of type `T`, or
2. go into an infinite loop.

Types for Arith

```
T ::= Bool
    | Nat
```

Typing rules for Arith
(in-class)

Is our type system "good"?

- Does it catch "bad" programs?
- Are there "good" programs that it prevents us from running?
- And what do we mean by "good" and "bad"?

Type Safety

If an expression *typechecks*, then it won't "get stuck". Either:

- the expression is a value
- an evaluation rule reduces the expression to a different expression

Safety = Progress + Preservation

- **Progress**: A well-typed expression won't get stuck.
- **Preservation**: A well-typed expression won't change its type during evaluation.

Type Safety, Formally

- Progress theorem:
  Suppose `e:T`. Then either
  1. `e` is a value
  2. There exists an `e'` such that `e -> e'`
- Preservation theorem:
  Suppose `e:T` and `e -> e'`.
  Then `e':T`.