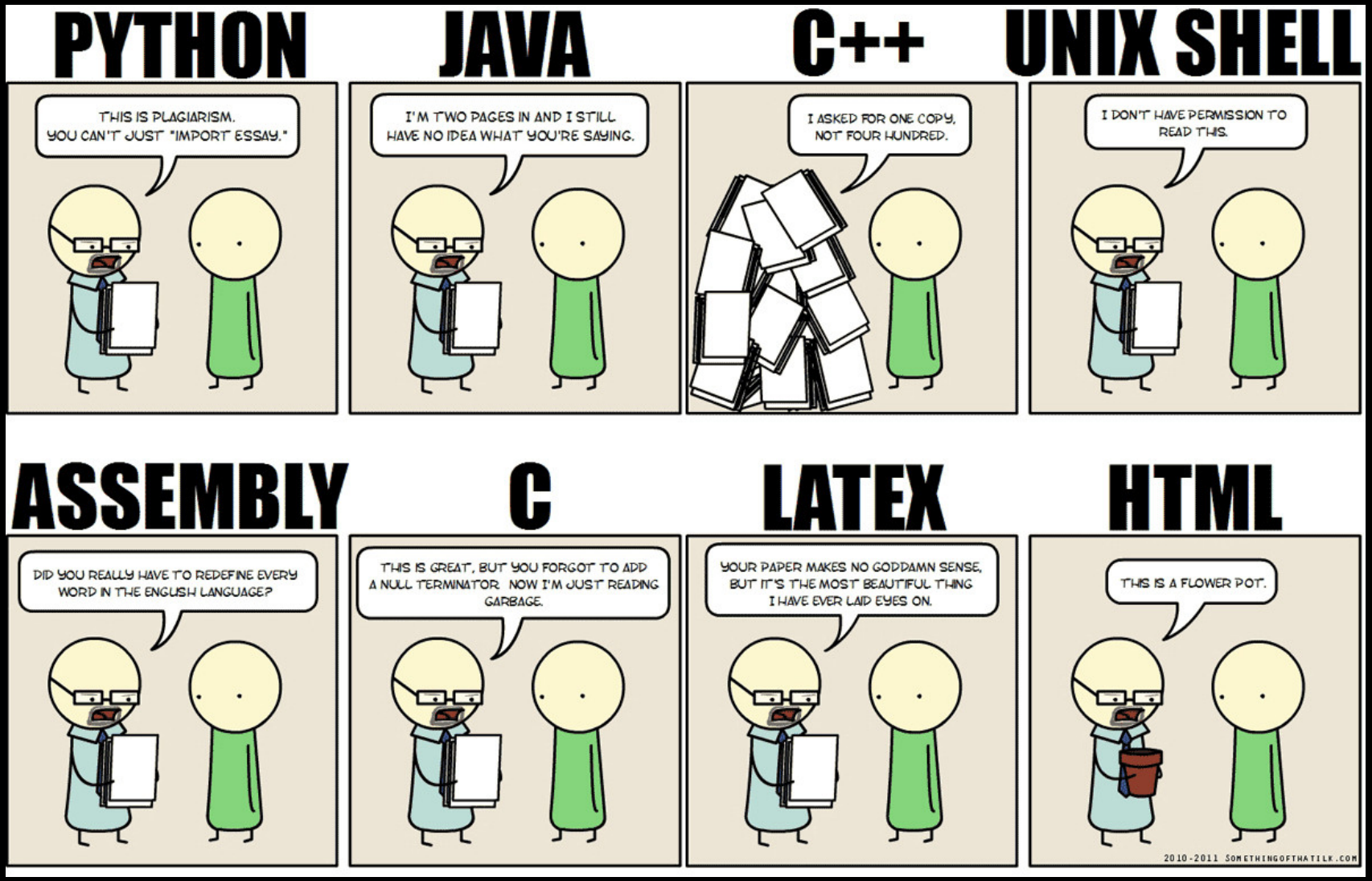
When You Write Your Essays in Programming Languages



http://www.somethingofthatilk.com/index.php?id=135

Office hours

- No office hours 9/7 or 9/11
- Additional office hours on 9/15 11am-1pm

CS 252:

Advanced Programming Language Principles

Operational Semantics

Prof. Tom Austin

San José State University

Lab Review

(in-class)

Why do we need formal semantics?

Everyone knows what an if statement does, right?

```
if true then
   x = 1
else
   x = 0
```

At the end of this code snippet, the value of x will be 1

Everyone knows what an if statement does, right?

```
if false then
   x = 1
else
   x = 0
```

At the end of this code snippet, the value of x will be 0

Everyone knows what an if statement does, right?

```
if 0 then
   x = 1
else
   x = 0
```

Will x be set to 0, like in C/C++?

Will x be set to 1, like in Ruby?

Or will it be an error, like in Java?

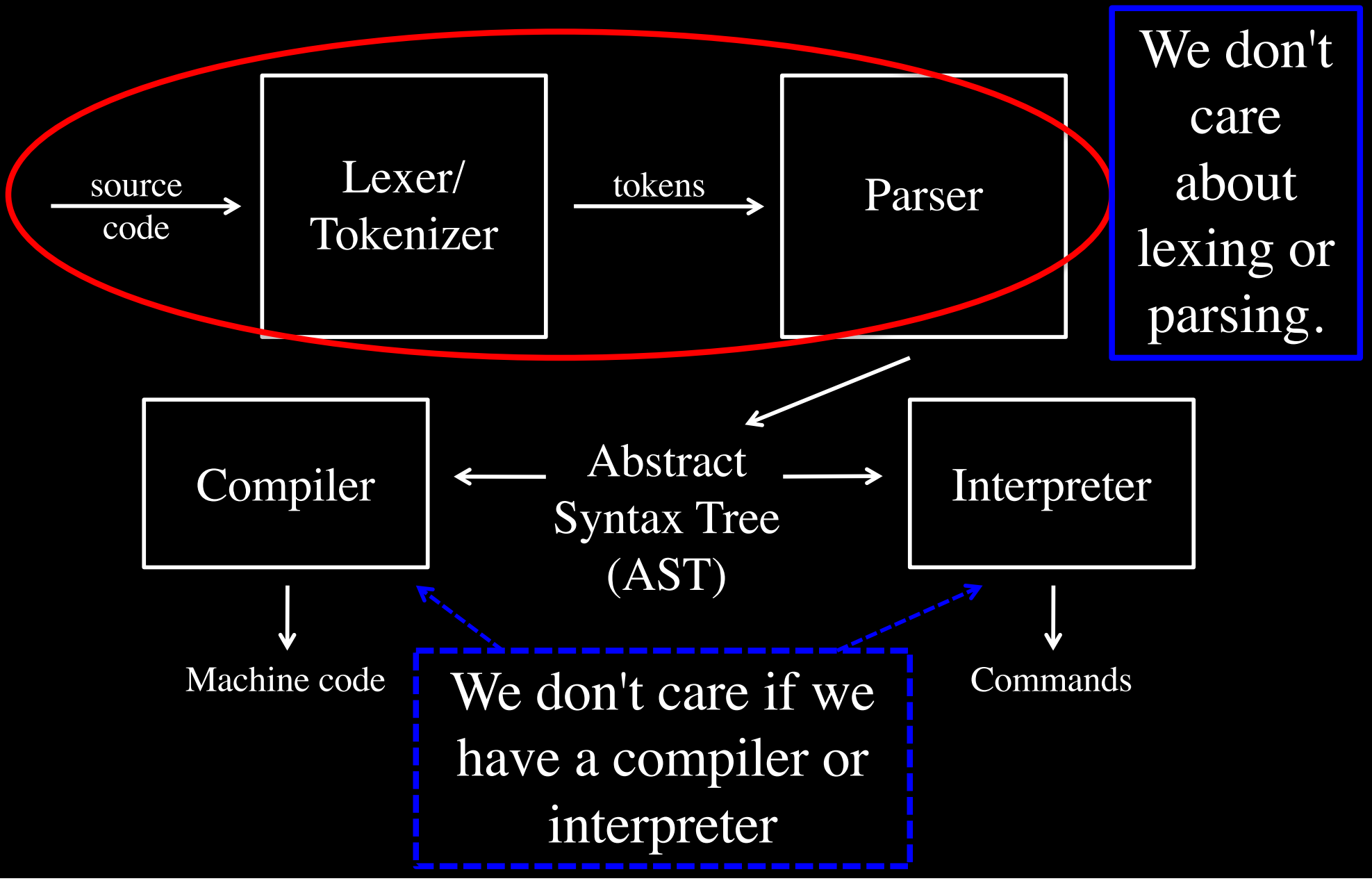
Everyone knows what an if statement does, right?

```
x = if true
      then 1
      else 0
```

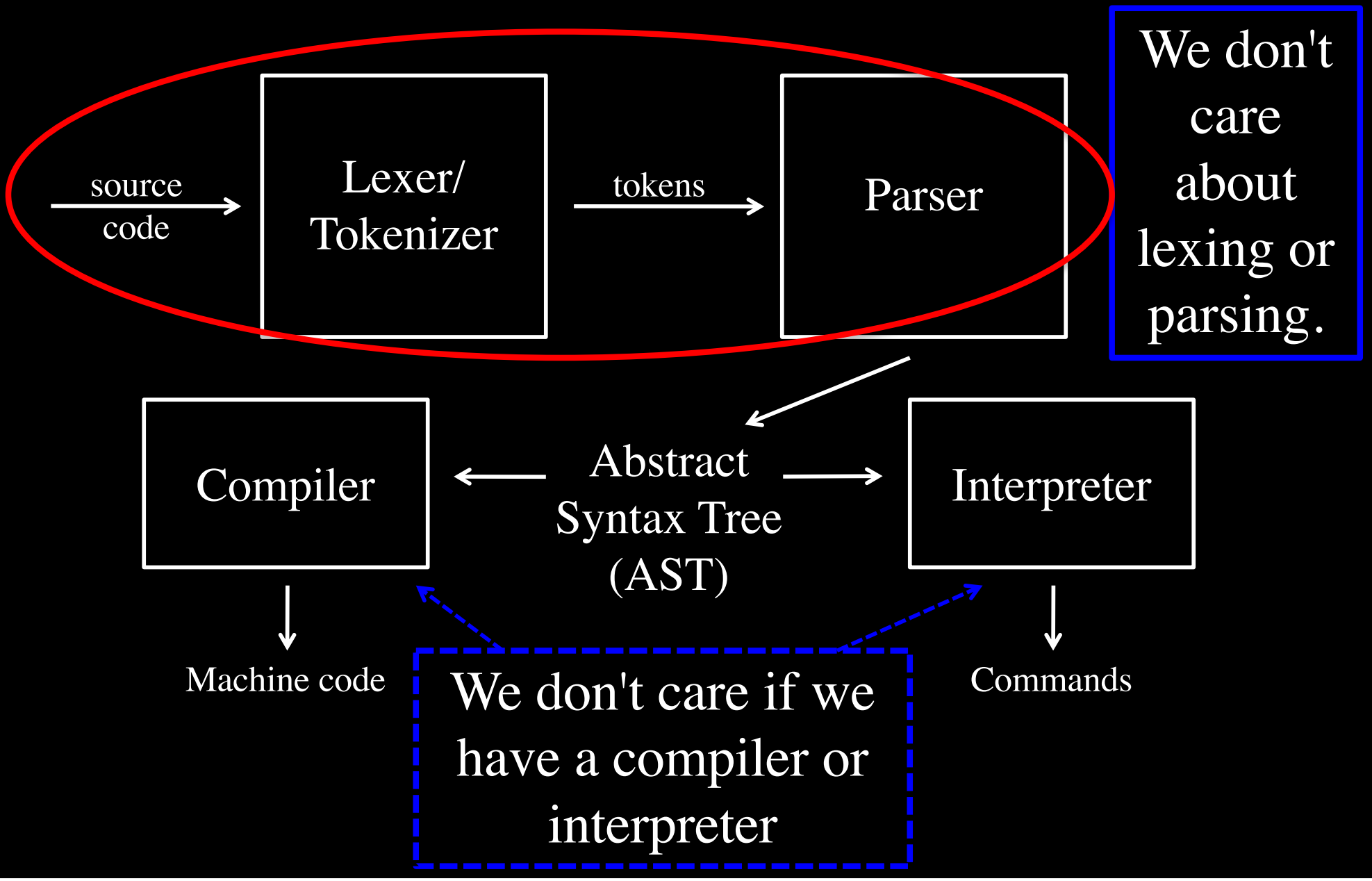
Is assignment valid or an error?

Formal semantics define how a language works *concisely and with minimal ambiguity*.

A Review of Compilers



A Review of Compilers



Abstract Syntax Tree (AST)

ASTs are the key to understanding a language

Bool* Language

```
e ::=
    true
  | false
  | if e
      then e
      else e
```

expressions:

constant true

constant false

conditional

Despite appearances, these are really ASTs

Values in Bool*

`v`	`::=`	*values:*
	`true`	constant true
`|`	`false`	constant false

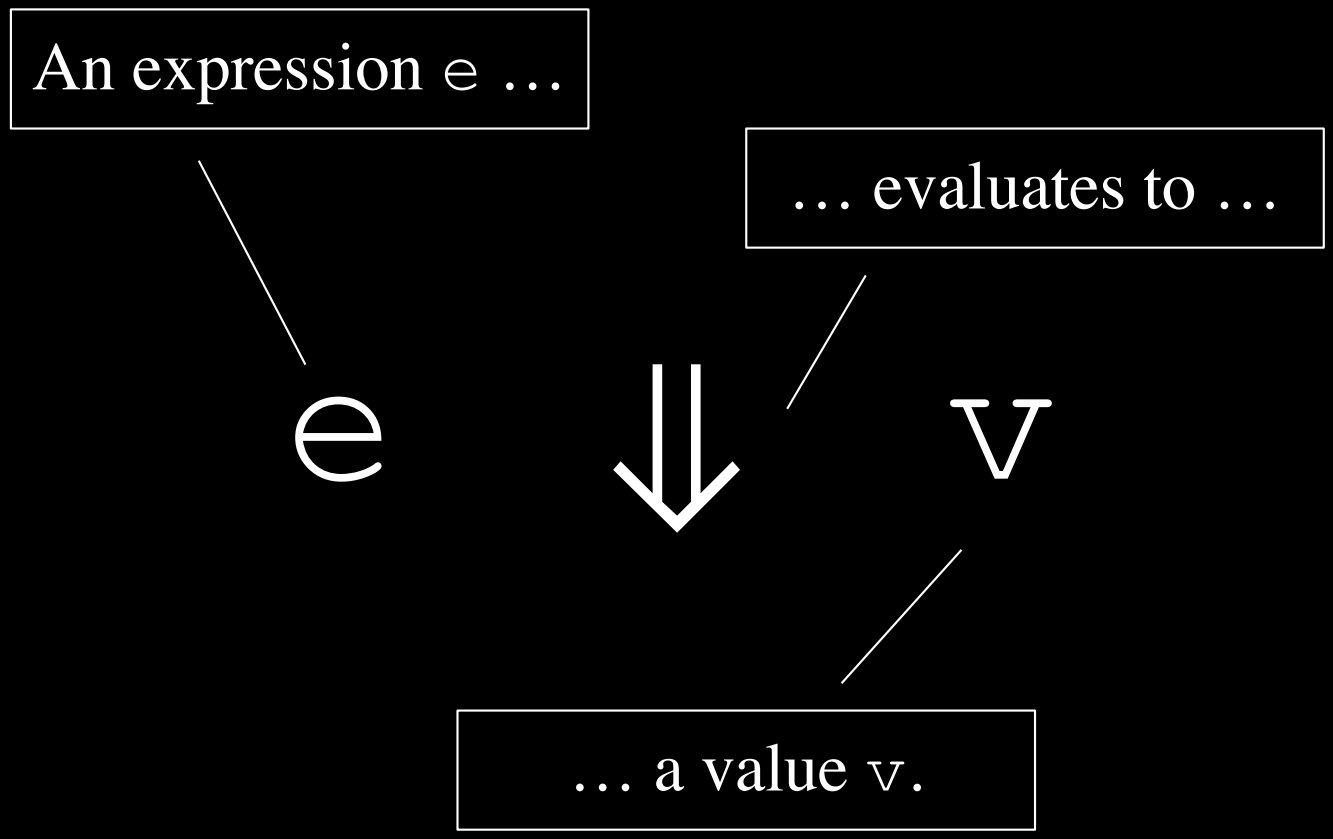
Formal Semantic Styles

- Operational semantics
  - Big-step (or "natural")
  - Small-step (or "structural")
- Axiomatic semantics
- Denotational semantics

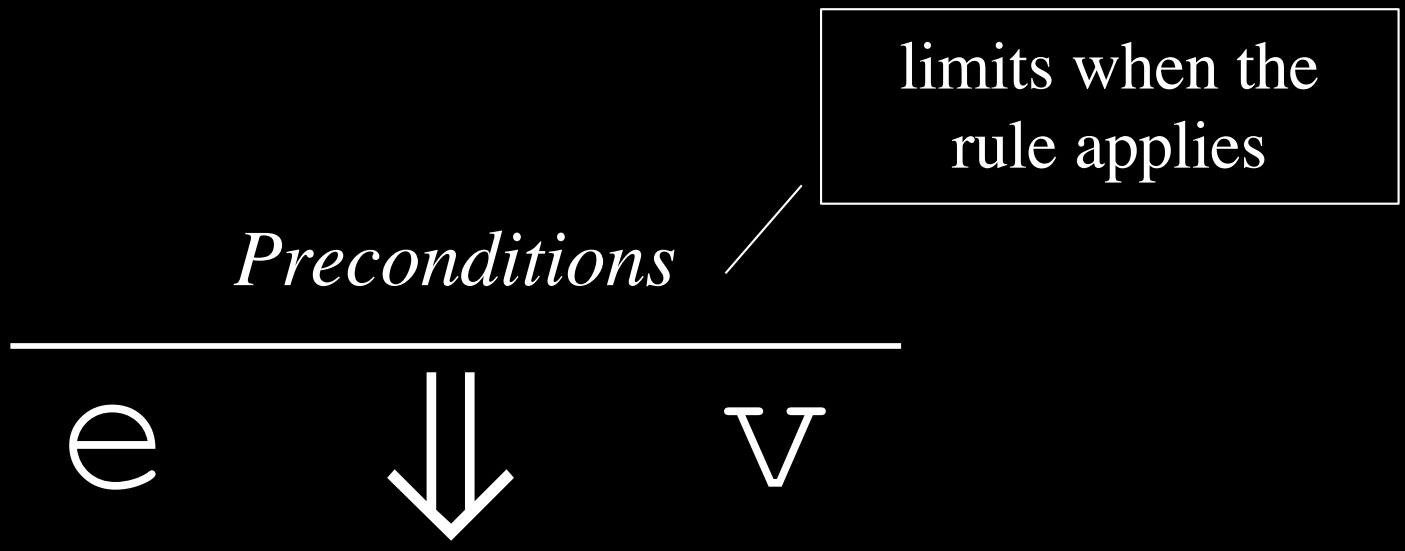
Formal Semantic Styles

- **Operational semantics**
  - **–Big-step (or "natural")**
  - –Small-step (or "structural")
- Axiomatic semantics
- Denotational semantics

Big-Step Evaluation Relation



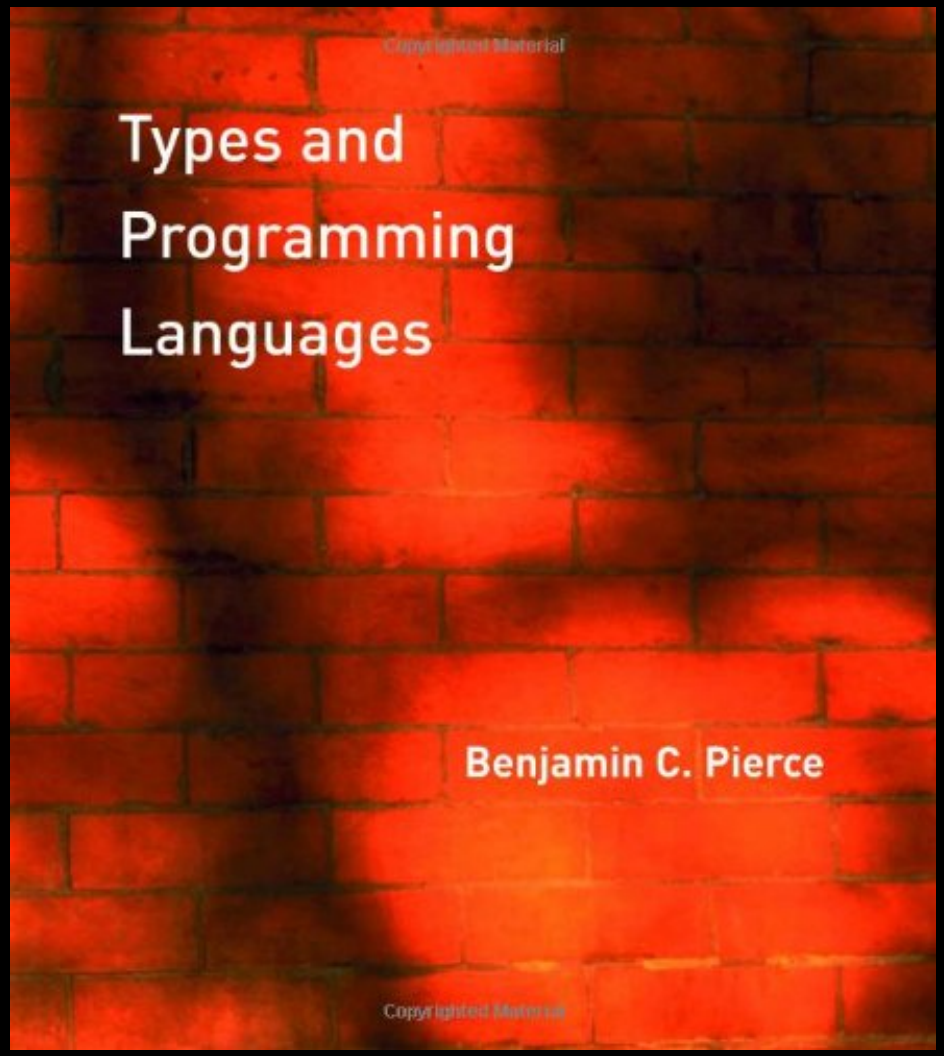
Big-Step Evaluation Relation



TAPL

Excellent reference for type systems and operational semantics.

Available at library.



Big-step semantics for Bool*

B-IfTrue

$$\frac{e1 \Downarrow \texttt{true} \qquad e2 \Downarrow v}{\texttt{if } e1 \texttt{ then } e2 \texttt{ else } e3 \Downarrow v}$$

B-IfFalse

$$\frac{e1 \Downarrow \texttt{false} \qquad e3 \Downarrow v}{\texttt{if } e1 \texttt{ then } e2 \texttt{ else } e3 \Downarrow v}$$

B-Value

$$\frac{}{v \Downarrow v}$$

Bool* big-step example

$$\dfrac{\dfrac{\dfrac{}{\texttt{true} \Downarrow \texttt{true}} \quad \dfrac{}{\texttt{false} \Downarrow \texttt{false}}}{\texttt{if true then false else true} \Downarrow \texttt{false}} \quad \dfrac{}{\texttt{false} \Downarrow \texttt{false}}}{\texttt{if (if true then false else true) then true else false} \Downarrow \texttt{false}}$$

Converting our rules into code

(in-class)

Language extension: numbers

Users demand a new feature – numbers!
We will add 3 new features:

- Numbers, represented by `n`
- `succ`, which takes a number and returns the next highest number.
- `pred`, which takes a number and returns the next lowest number.

BoolNum* Language

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

Let's extend our semantics to handle these new language constructs

Extended values and semantic rules

(in-class)

Literate Haskell

- Files use .lhs extension (rather than .hs)
- Code lines begin with >
- All other lines are comments

```
-- Regular .hs      Literate Haskell
-- source file      src file (.lhs)

foo x = 1           > foo x = 1
  + (foo (x - 1))   >   + (foo (x - 1))
```

Lab 2: Write a Bool* Interpreter

- Starter code is available at http://cs.sjsu.edu/~austin/cs252-fall17/labs/lab2/interp.lhs
- Part 1: Complete evaluate function
- Part 2: Extend Bool* with `0`, `succ`, and `pred`

Example: Information Flow Analysis

- Goal: prevent secrets from leaking
  - E.g. protect credit card info
  - Attacker might control some code
- We will
  - Mark sensitive data
  - Keep track of which data is secret

SecretKeeper Language

```
e ::= true | false | n
    | if e then e else e
    | succ e
    | pred e
    | secret e
```

Semantics assignment 1

- Write the operational semantics for SecretKeeper
- Hint: values need to be marked either secret or public. Your evaluation relation might be

$$e \Downarrow v^{\texttt{label}}$$