CS 152: *Programming Language Paradigms*

Modules, Structs, Hashes, and Operational Semantics

Prof. Tom Austin

San José State University

Modules

Review Modules from HW 1

(in-class)

How do we organize code in Java?

- **Packages** provide units of code
- Keywords specify method access

In Java, keywords specify access

- **`public`**
- **`protected`**
- no keyword (package access)
- **`private`**

Organizing Code in Racket

- *Exports* specify public values:
```
(provide big-add)
```

- *Imports* specify code dependencies:
```
(require "big-num.rkt")
```

Structs and Hashes

Structs

- *Structures* allow us to create more sophisticated data structures:
  `(struct` *`name`* `(`*`field1 field2`* `…` `))`
- Once we have a structure, we can *destructure* it with the **`match`** keyword to get at the contents.
- <Example in class>

Hashes

- Hashes are maps of key/value pairs.
- *Unlike* Java, hashes are immutable.
- <example in class>

Formal Semantics

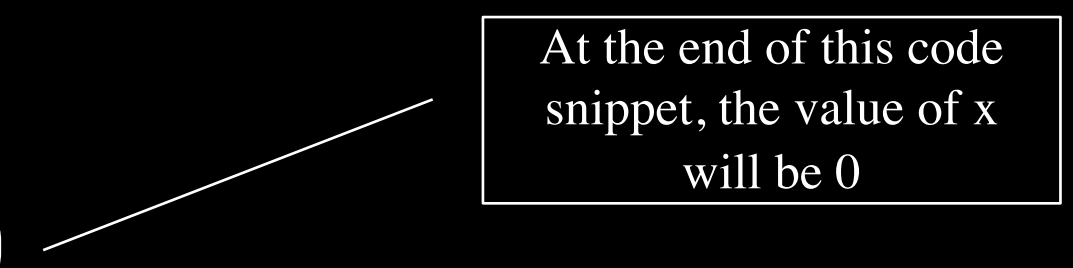
Why do we need formal semantics?

Everyone knows what an if statement does, right?

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

Everyone knows what an if statement does, right?

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

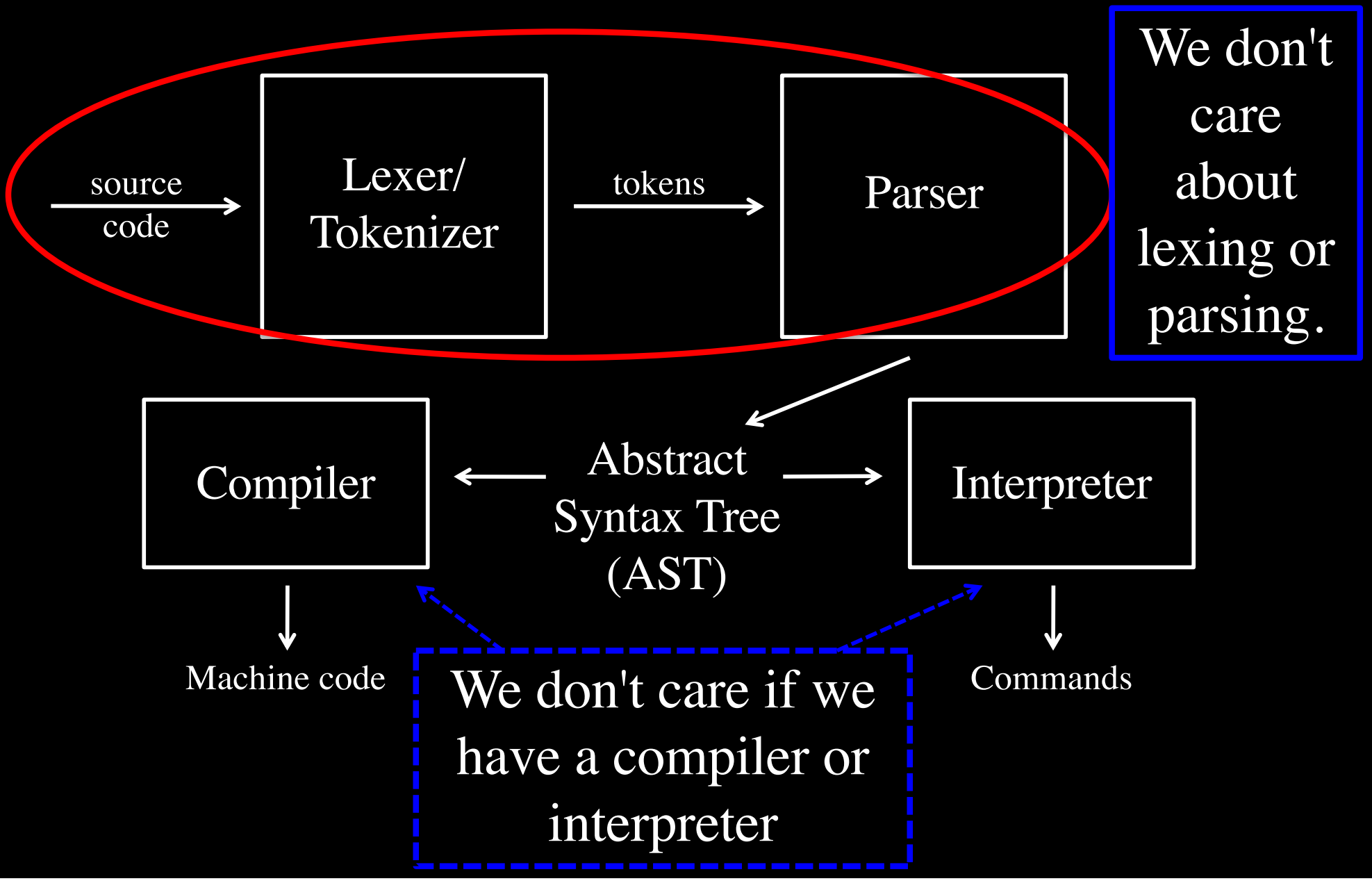
Everyone knows what an if statement does, right?

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

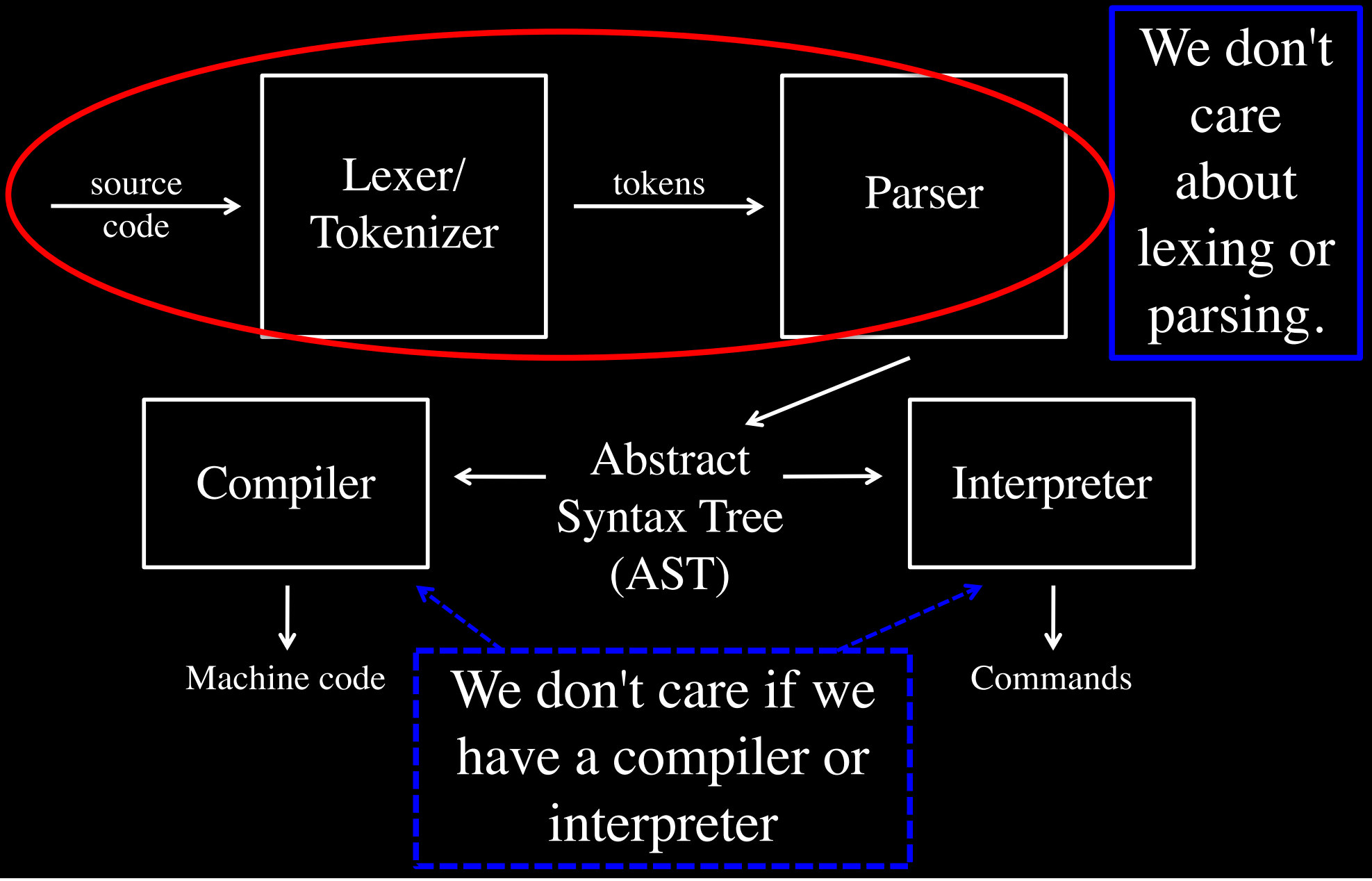
Is assignment valid or an error?

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

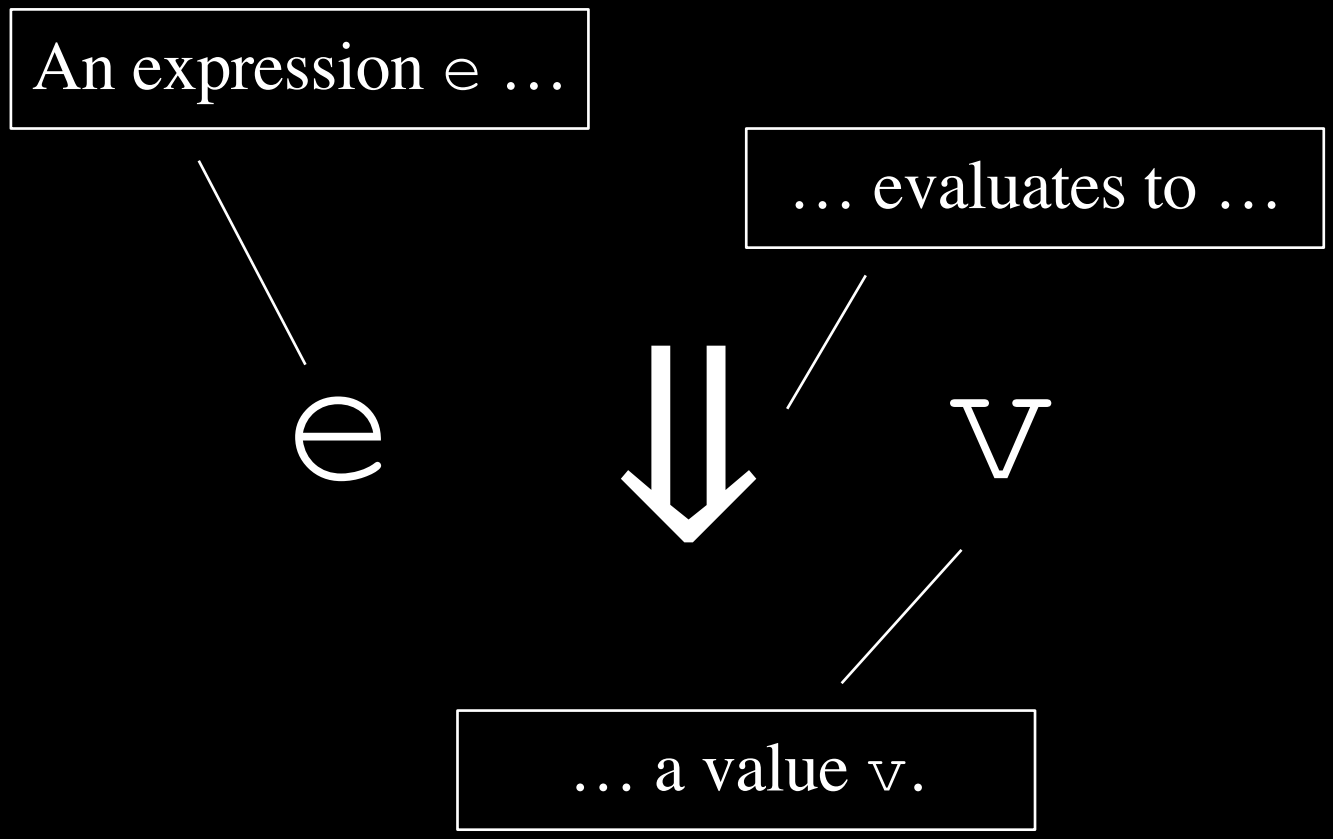
Formal Semantic Styles

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

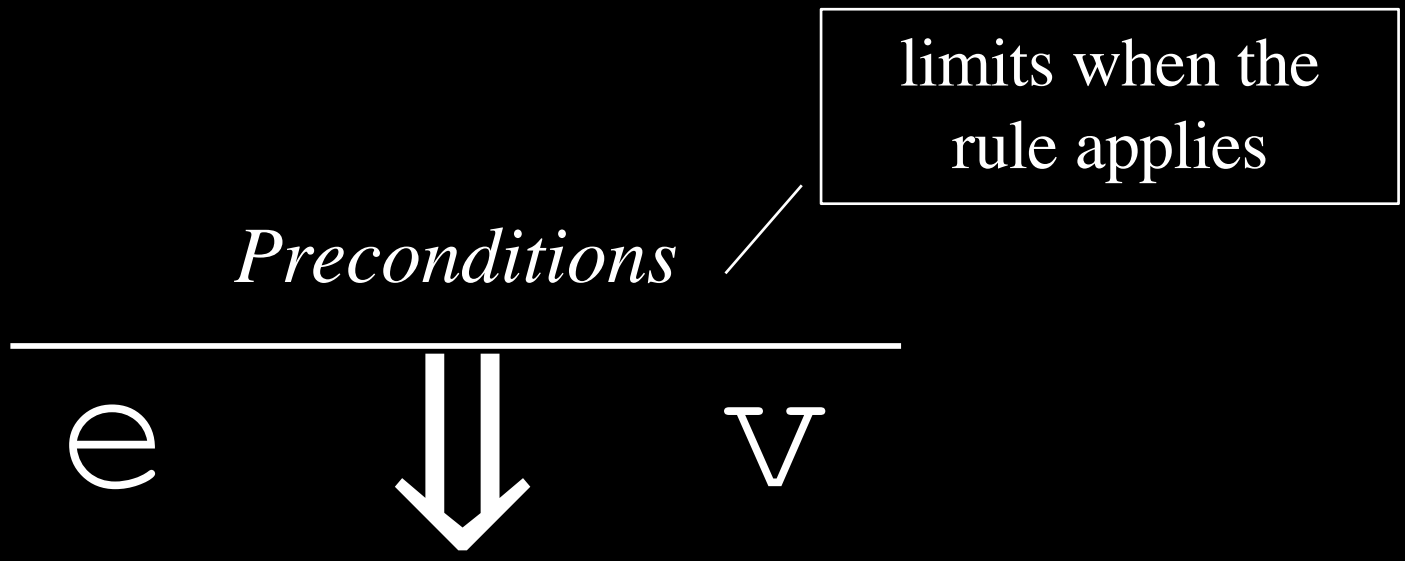
Formal Semantic Styles

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

Big-Step Evaluation Relation



Big-Step Evaluation Relation



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

$$\frac{\texttt{true} \Downarrow \texttt{true} \quad \texttt{false} \Downarrow \texttt{false}}{\texttt{if true then false else true} \Downarrow \texttt{false}} \qquad \texttt{false} \Downarrow \texttt{false}$$

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

Converting our rules into code

(in-class)

Bool* 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.

Extended Bool* 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

Lab: Write a Bool* Interpreter

- Starter code is available on the course website
- Extend Bool* with numbers, `succ`, and `pred`

Adding State to Semantics

SpartanLang

```
e ::= !x                        dereferencing
    | v                         values
    | x:=e                      assignment
    | e;e                       sequence
    | e op e                    binary operations
    | if e then e               conditionals
        else e end
    | while e do e end          while loops
```

SpartanLang (continued)

```
v  ::= i          integers
     | b          booleans

op  ::= +   |  -    binary operators
      | \   |  *
      | <   |  >
      | <=  |  >=
```

Bool* vs. SpartanLang evaluation

Bool* relation:

$$e \Downarrow v$$

SpartanLang relation:

$$e, \sigma \Downarrow v, \sigma'$$

A "store", represented by the Greek letter sigma

The Store

- A mapping of references to values
- Roughly analogous to the heap in Java

Key store operations

- $\sigma(x)$
  - get the value for reference x.
- $\sigma[x := v]$
  - create a copy of store σ, except …
  - reference x has value v.

HW 2: Write an Interpreter for SpartanLang.

Details in Canvas