CS 252:

Advanced Programming Language Principles

Operational Semantics

Prof. Tom Austin

San José State University

Why do we need formal semantics?

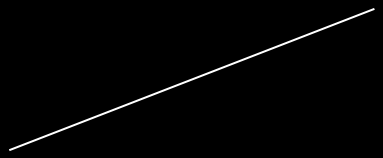
Everyone knows what an
if statement does, right?

```
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?

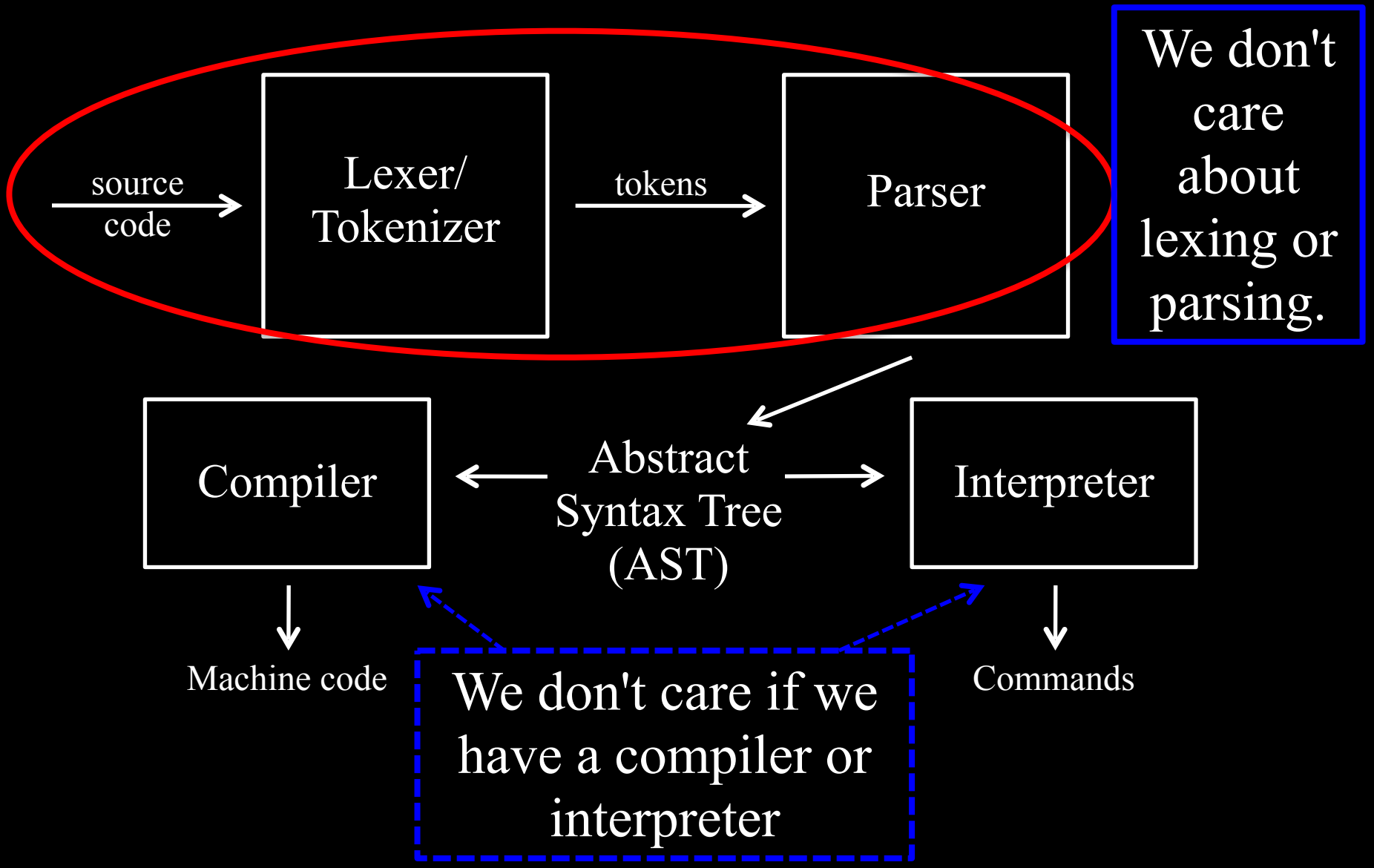
Everyone knows what an
if statement does, right?

```
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



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

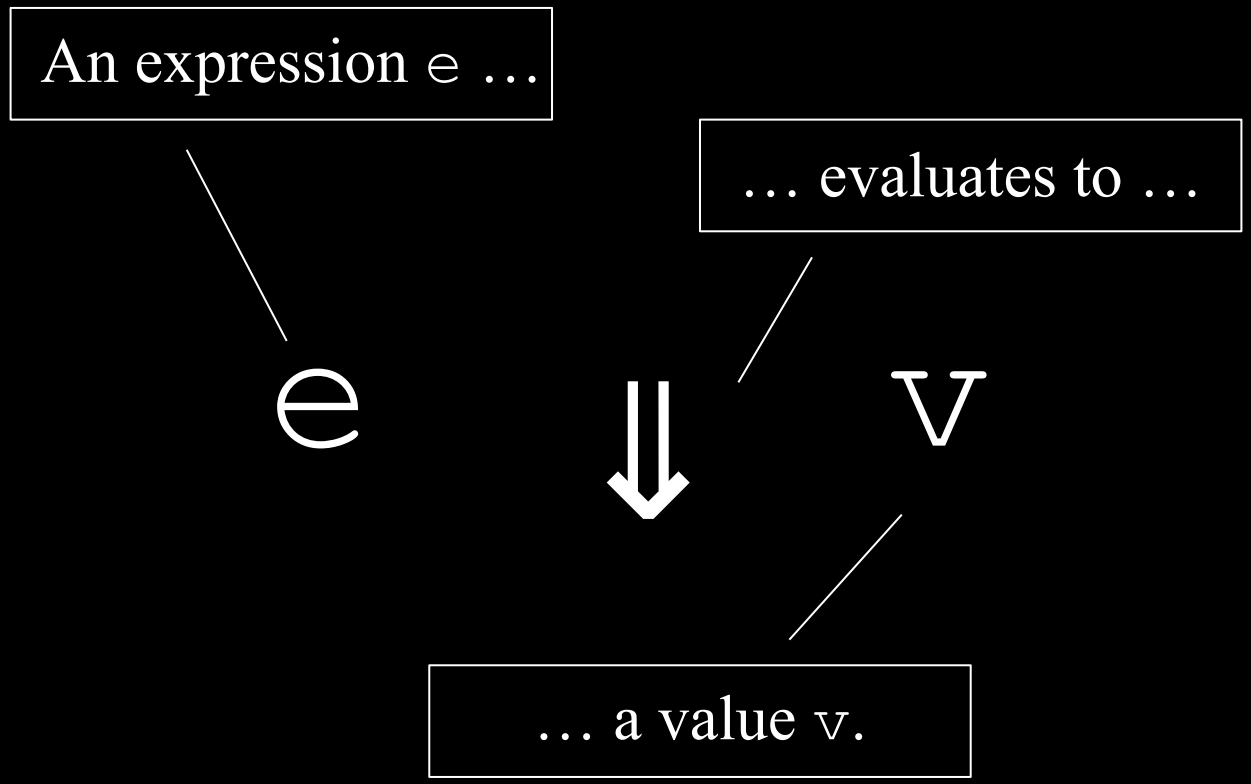
Formal Semantic Styles

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

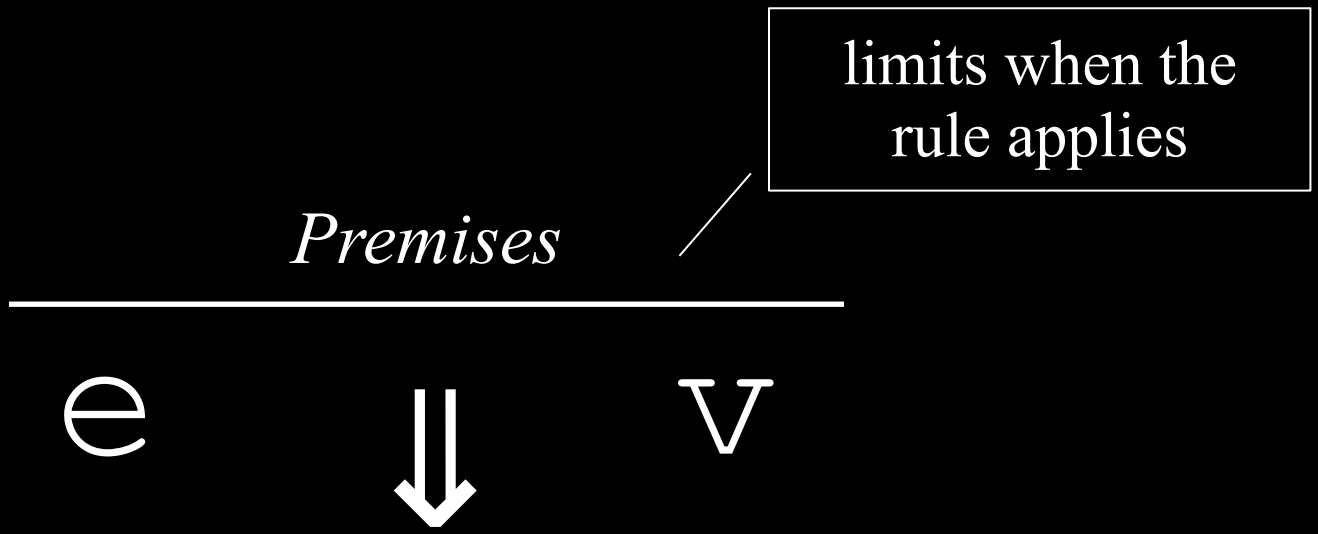
Formal Semantic Styles

- **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-True

$$\texttt{true} \Downarrow \texttt{true}$$

B-False

$$\texttt{false} \Downarrow \texttt{false}$$

B-IfTrue

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

B-IfFalse

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

Big-step semantics for Bool*

B-Value

$$v \Downarrow v$$

More concise version

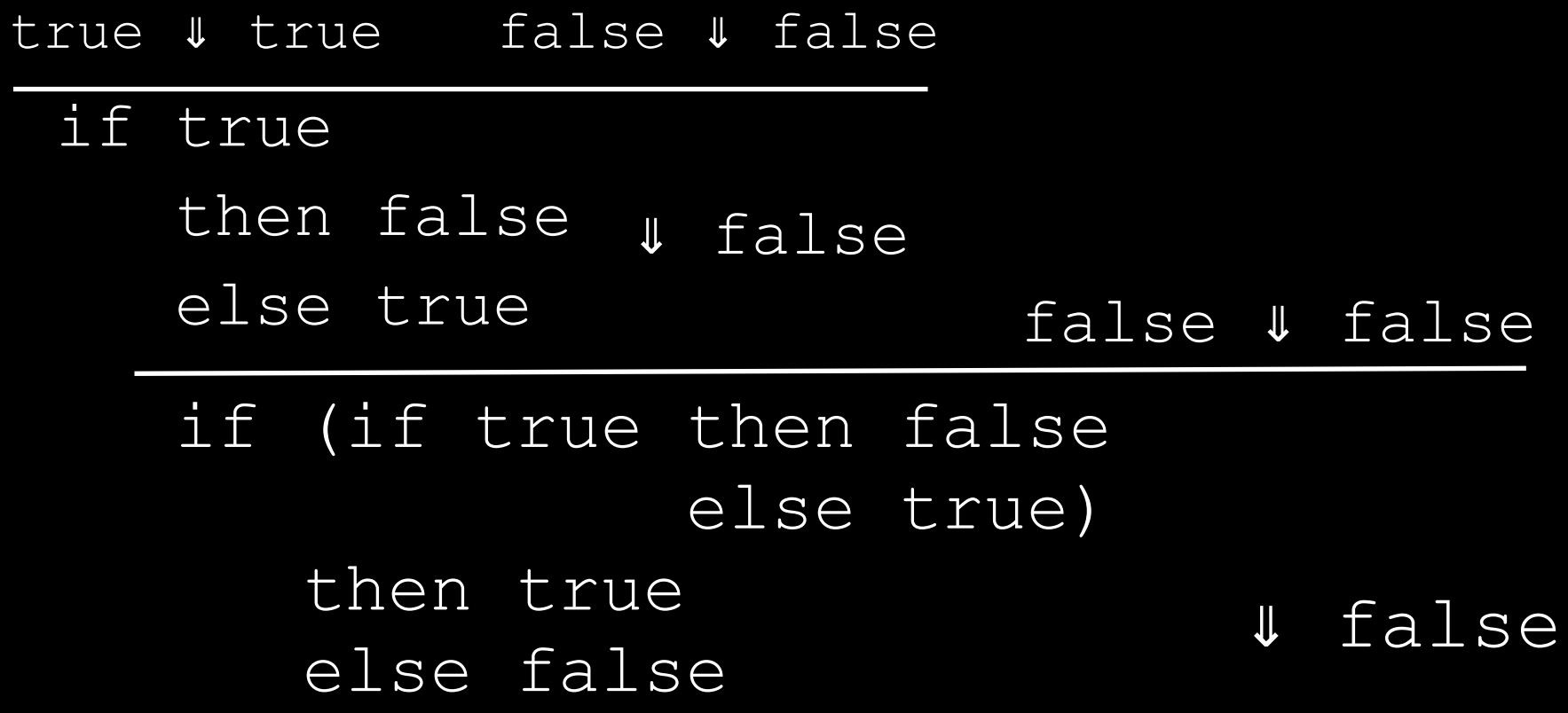
B-IfTrue

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

B-IfFalse

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

Bool* big-step example



Converting our rules into code

(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))
```

Haskell tip: case/of

```
maxNum :: [Integer] -> Integer
maxNum lst = case lst of
   [] -> error "Empty list"
   [x] -> x
   (x:xs) -> let mx = maxNum xs in
     if x > mx then x else mx
```

Lab 3, Part 1: Write a Bool* Interpreter

Starter code is available on course website.

Complete `evaluate` function.

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

Lab 3, Part 2: Extend Bool* Interpreter

Extend Bool* interpreter with numbers, `succ`, and `pred`.

Update semantics.tex from the course website and your Haskell interpreter from part 1. Your LaTeX and your Haskell code should match.

Extended values and semantic rules

(in-class)

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
```

Sample Program

Assume `cc` is a credit card.

If this code:

```
succ (secret cc)
```

produces `41111111111111112L`,

then what is `cc`?

And who can read it?

Semantics assignment 1

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

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