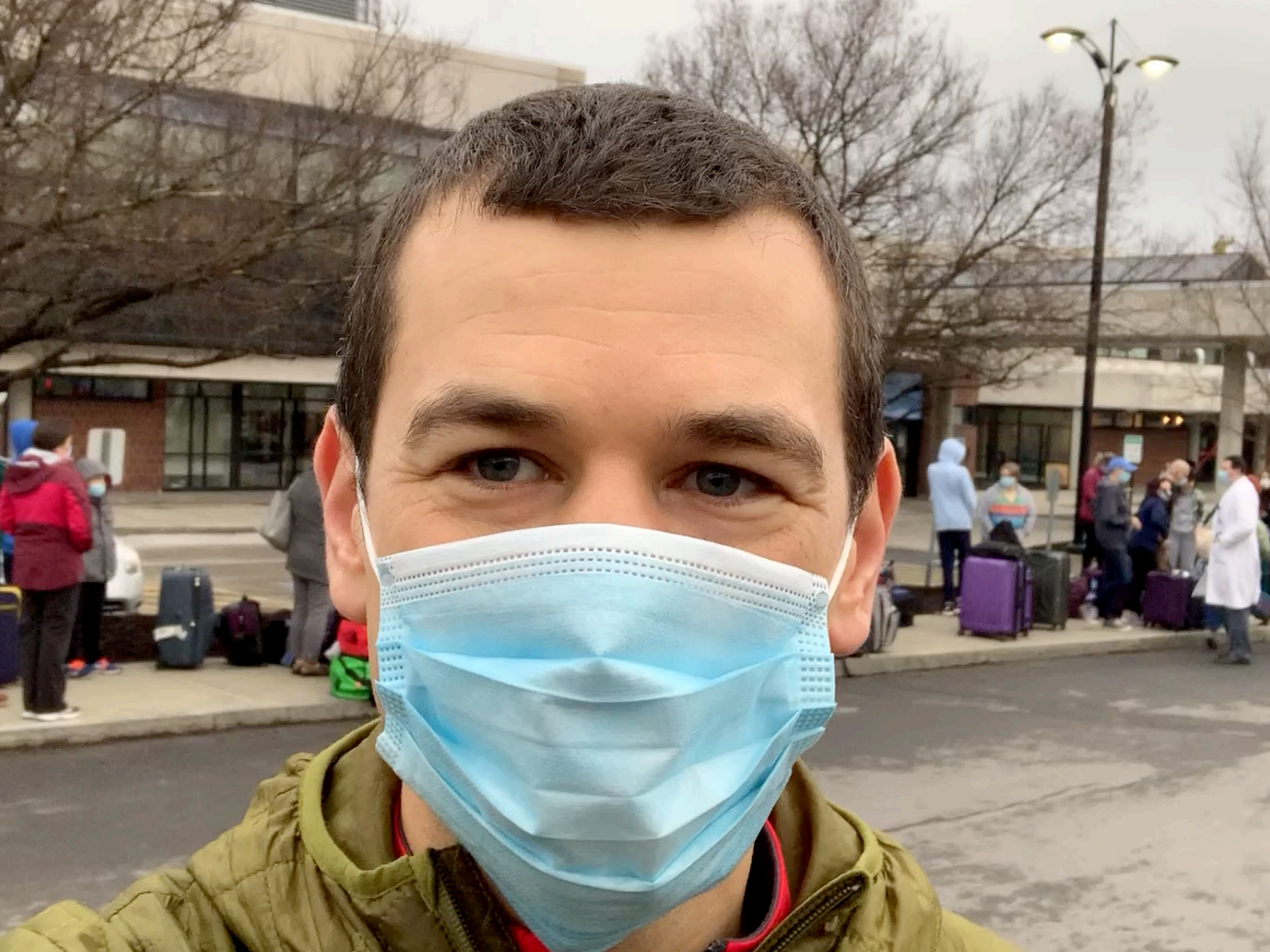
CS 3110

The Substitution Model

Nate Foster

Spring 2020

Today's scene: Cayuga Medical Center



Review

Previously in 3110:

- interpreter for tiny calculator language
- lexing into tokens
- parsing into abstract syntax tree (AST)
- evaluation based on single steps

Today:

- define evaluation formally
- extend to a bigger language: let expressions
- substitution model: a way to think about evaluation of `let`

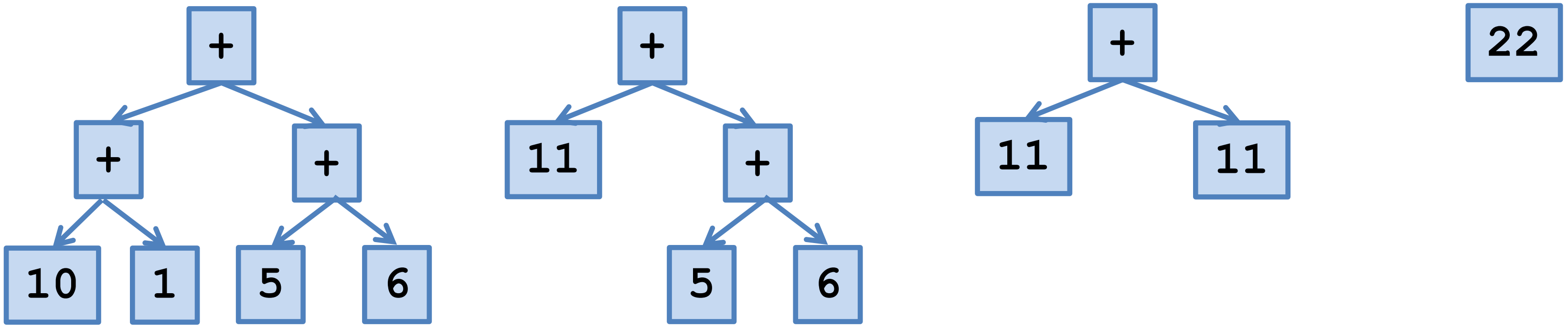
Review: calculator language BNF

```
e ::= i
    | e1 bop e2
    | ( e )

bop ::= + | *

i ::= integers
```

Demo

Review: evaluation strategy

- An expression e takes a single *step* to a new expression e’ by simplifying some subexpression
- Expression keeps stepping until it reaches a *value*
- Values never step further



Demo

FORMAL DYNAMIC SEMANTICS

$$\texttt{e} \longrightarrow \texttt{e'}$$

single-step relation

the `step` function we implemented

$$\mathbf{v} \not\rightarrow$$

values never step

$$\texttt{e} \longrightarrow^{*} \texttt{e'}$$

multi-step relation

related to the `eval` function we implemented

Inductively defined relation

`e1 + e2` → `e1' + e2`

if `e1` → `e1'`

`v1 + e2` → `v1 + e2'`

if `e2` → `e2'`

`v1 + v2` → `i`

if `i` *is the result of primitive operation* `v1 + v2`

PART II: LET EXPRESSIONS

Let expressions

```
e  ::= i
     | e1 bop e2
     | ( e )
     | x
     | let x = e1 in e2

bop ::= + | *
i  ::= integers
x  ::= identifiers
```

Let semantics

```
let x = e1 in e2 → let x = e1' in e2
   if e1 → e1'

let x = v1 in e2 →
   e2 with v1 substituted for x
```

$$e\{v/x\}$$

means e with v substituted for x

Let semantics

`let x = e1 in e2` ⟶ `let x = e1' in e2`

  if `e1` ⟶ `e1'`

`let x = v1 in e2` ⟶ `e2{v1/x}`

Demo

Defining substitution: the easy parts

Nothing to do for integers:

$$i \{v/x\} = i$$

Just keep going through operations:

$$(e1 + e2) \{v/x\} = (e1 \{v/x\}) + (e2 \{v/x\})$$

Variables are where substitution really happens:

$$x \{v/x\} = v$$

$$y \{v/x\} = y$$

Defining substitution: let

(let y = e1 in e2) {v/x}

=

let y = (e1 {v/x}) in (e2 {v/x})

(let x = e1 in e2) {v/x}

=

let x = (e1 {v/x}) in e2

Do substitute in binding.
e.g.,
let x = 1 in
(let y = x in y)

Stop substituting in body.
e.g.,
let x = 1 in
(let x = 2 in x)

gets even trickier in the presence of functions: see textbook re. capture-avoiding substitution

If expressions

```
e ::= x | i | b
    | e1 bop e2
    | let x = e1 in e2
    | if e1 then e2 else e3

bop ::= + | * | <=
```

If semantics

```
if e1 then e2 else e3 → if e1' then e2 else e3
   if e1 → e1'

if true then e2 else e3 → e2

if false then e2 else e3 → e3
```

Demo

Upcoming events

- [last night] R5 due
- [Friday] Honor Code Quiz Due
- [Friday] A4 due

This is not a substitute.

THIS IS 3110