

S

Semantics Intro

CIS700 — Fall 2022

Kris Micinski



Why Semantics?

- Semantics let us define what a language **means**
- How to do this?
 - **Operationally** — define an “interpreter” for the language, semantics is whatever the interpreter produces.
 - Interpreter must be small to be trustworthy—otherwise hard to trust
 - **Denotationally** — define a mapping from programs to mathematical objects (e.g., functions from/to probability distributions)

Operational Semantics

- We will focus on two kinds of operational semantics
 - “**Big Step**” define a *recursive* interpreter, which produces *proofs* of its correctness by recursively walking over terms / judgements

- Output is “a big tree”

- “**Small Step**”

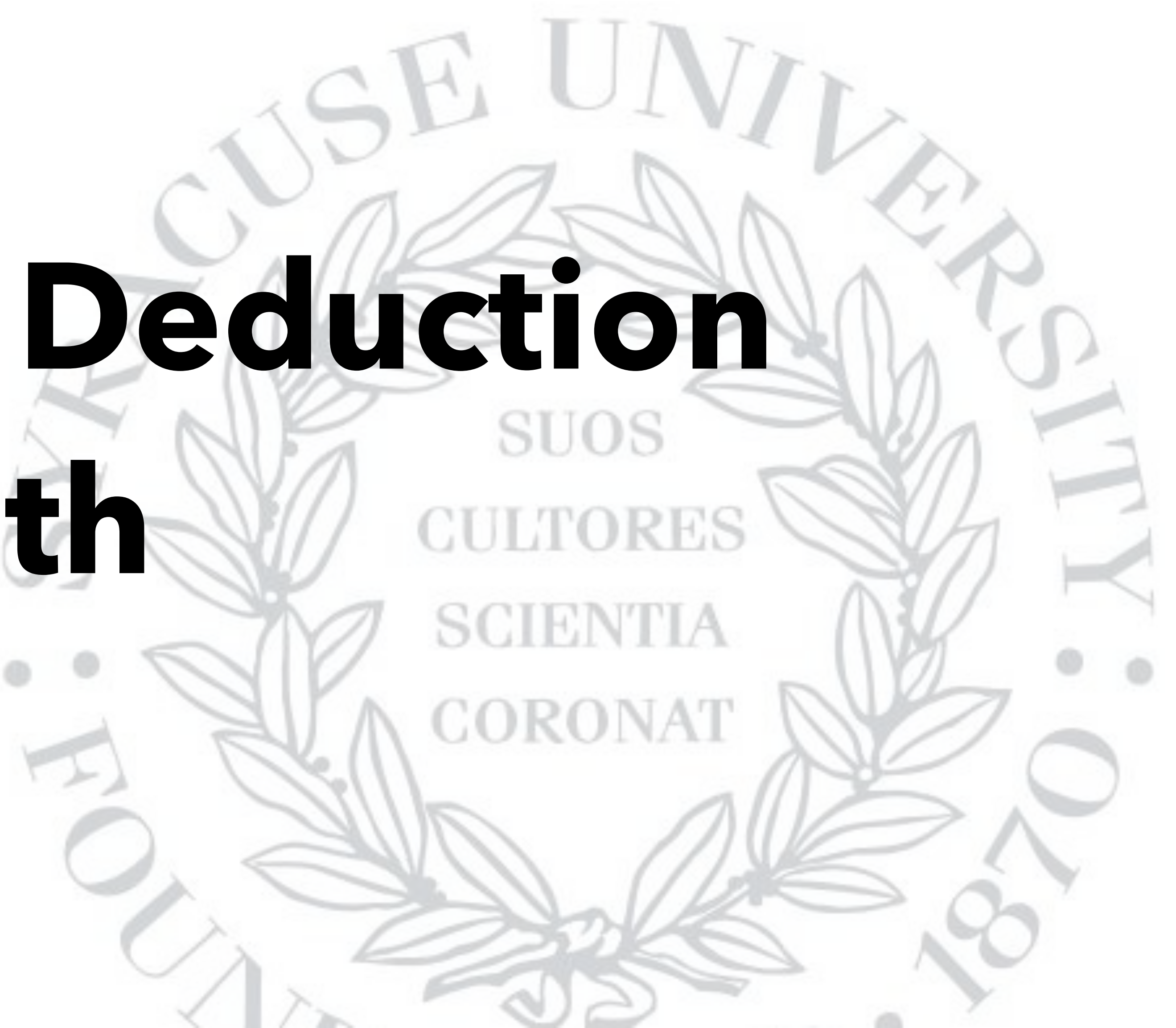
$$\frac{
 \frac{
 \frac{(1 \ \emptyset \Downarrow (S \ U))}{((if0 \ 1 \ (plus \ 1 \ 1) \ 0) \ \emptyset \ \Downarrow \ 0)}
 \quad
 \frac{(0 \ \emptyset \ \Downarrow \ 0)}{((if0 \ 1 \ (plus \ 1 \ 1) \ 0) \ \emptyset \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 \quad
 \frac{
 \frac{(x \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0) \quad ((let \ y \ x \ x) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 }{((if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x)) \ (\mapsto \ \emptyset \ x \ 0) \ \Downarrow \ 0)}
 }{((let \ x \ (if0 \ 1 \ (plus \ 1 \ 1) \ 0) \ (if0 \ x \ (let \ y \ x \ x) \ (let \ z \ x \ x))) \ \emptyset \ \Downarrow \ 0)}$$

- Output is a **sequence** of *states*; semantics is transitive closure of transition relation
- We will discuss both of these for a dirt-simple language, IfArith

S

Natural Deduction for IfArith

Kris Micinski



In this lecture, we'll introduce **natural deduction**

Natural deduction is a mathematical formalism that helps ground the ideas in metacircular interpreters

Natural deduction first used in mathematical logic, to specify **proofs** using inductive data

We will use natural deduction as a framework for specifying semantics of various languages throughout the course

Introduction Rules

$$\frac{\begin{array}{c} \frac{}{\vdash^N A} u \\ \vdots \\ \vdash^N B \end{array}}{\vdash^N A \supset B} \supset I^u$$

$$\frac{\begin{array}{c} \frac{}{\vdash^N A} u \\ \vdots \\ \vdash^N p \end{array}}{\vdash^N \neg A} \neg I^{p,u}$$

$$\frac{\vdash^N [a/x]A}{\vdash^N \forall x.A} \forall I^a$$

Elimination Rules

$$\frac{\vdash^N A \supset B \quad \vdash^N A}{\vdash^N B} \supset E$$

$$\frac{\vdash^N \neg A \quad \vdash^N A}{\vdash^N C} \neg E$$

$$\frac{\vdash^N \forall x.A}{\vdash^N A} \forall E$$

When we specify the semantics of a language using natural deduction, we give its semantics via a set of **inference rules**

Rules read: if the thing on the **top** is true, then the thing on the **bottom** is also true.

This rule says: "if c is an integer
(mathematically: $c \in \mathbb{Q}$), then c evaluates to c ."

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c}$$

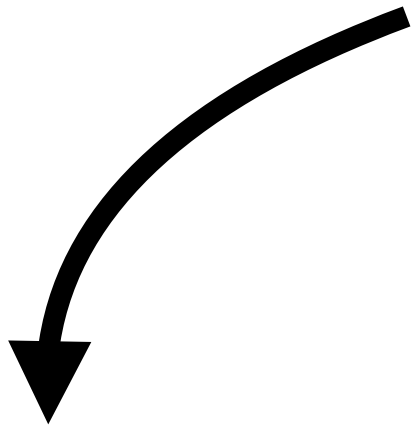
Note: the notation $e \Downarrow v$ is read "e evaluates to v."

Some rules will have more than one **antecedent** (thing on the top).

You read these: “if the first thing, and second thing, and ... are **all** true, then the thing on the bottom is true.”

$$\mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

"If $e_0 \Downarrow n_0$, and $e_1 \Downarrow n_1$, and $n' = n_0 + n_1$, **then** I can say
(plus $e_0 e_1$) $\Downarrow n'$."



Plus : $\frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 e_1) \Downarrow n'}$

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

The natural deduction rule for **div** is similar

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1} \quad \mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

We have **two** rules for not

Natural Deduction Rules for IfArith

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1} \quad \mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

Question: Now that we have the rules, what can we do with them?

Answer: Use them to **formally prove** that some program calculates some result

Let's say I want to prove that the following program evaluates to 4:

```
(if (plus 1 -1) 3 4)
```

What rule could go here..?

$$\frac{???}{(\text{if } (\text{plus } 1 \text{ } - 1) 3 4) \Downarrow 4}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow n \quad n \neq 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

$$\frac{???}{(\text{if } (\text{plus } 1 \ - 1) \ 3 \ 4) \Downarrow 4}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow n \quad n \neq 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

$$\frac{???}{(\text{if } (\text{plus } 1 \ - 1) \ 3 \ 4) \Downarrow 4}$$

To apply a natural-deduction rule,
we must perform **unification**

**There can be no variables in the
resulting unification!**

$$\mathbf{If}_F : \frac{e_0 \Downarrow 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

$$\frac{(\text{plus } 1 \ -1) \Downarrow 0 \qquad 4 \Downarrow 4}{(\text{if } (\text{plus } 1 \ -1) \ 3 \ 4) \Downarrow 4}$$

We perform unification:

e_0 : (plus 1 -1), e_1 : 3

e_2 : 4, n' : 4

Not done yet, now we have to prove
these things

$$\frac{(\text{plus } 1 \text{ } - 1) \Downarrow 0 \quad 4 \Downarrow 4}{(\text{if } (\text{plus } 1 \text{ } - 1) \text{ } 3 \text{ } 4) \Downarrow 4}$$

Why can we say $4 \Downarrow 4$? Because of the **Const** rule

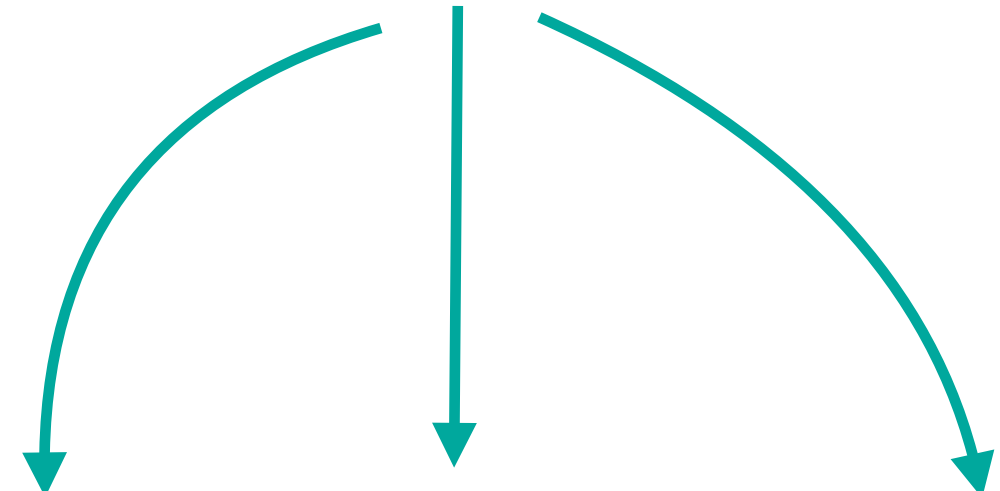
$$\frac{(\text{plus } 1 \text{ } - 1) \Downarrow 0 \quad \frac{4 \in \mathbb{Q}}{4 \Downarrow 4}}{(\text{if } (\text{plus } 1 \text{ } - 1) \text{ } 3 \text{ } 4) \Downarrow 4}$$

We're not done yet, because **plus** requires an antecedent:

$$\mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\frac{(\text{plus } 1 \ - \ 1) \Downarrow 0 \quad \frac{4 \in \mathbb{Q}}{4 \Downarrow 4}}{(\text{if } (\text{plus } 1 \ - \ 1) \ 3 \ 4) \Downarrow 4}$$

But we're **still** not done, because we need to finish these three



$$\begin{array}{r}
 1 \Downarrow 1 \quad -1 \Downarrow -1 \quad 1 + -1 = 0 \\
 \hline
 \text{(plus } 1 - 1) \Downarrow 0 \\
 \hline
 \text{(if (plus } 1 - 1) 3 4) \Downarrow 4
 \end{array}
 \qquad
 \begin{array}{r}
 4 \in \mathbb{Q} \\
 \hline
 4 \Downarrow 4
 \end{array}$$

Things that are simply true from algebra require no antecedents, we take them as "axioms."

$$\begin{array}{c}
 \frac{\frac{1 \in \mathbb{Q}}{1 \Downarrow 1} \quad \frac{-1 \in \mathbb{Q}}{-1 \Downarrow -1} \quad \frac{}{1 + -1 = 0}}{\text{(plus } 1 - 1) \Downarrow 0} \quad \frac{4 \in \mathbb{Q}}{4 \Downarrow 4} \\
 \hline
 \text{(if (plus } 1 - 1) 3 4) \Downarrow 4
 \end{array}$$

This is a complete proof that the program computes 4

$$\frac{\frac{1 \in \mathbb{Q}}{1 \Downarrow 1} \quad \frac{-1 \in \mathbb{Q}}{-1 \Downarrow -1} \quad \overline{1 + -1 = 0}}{\text{(plus 1 - 1) } \Downarrow 0} \quad \frac{4 \in \mathbb{Q}}{4 \Downarrow 4}$$

$$\text{(if (plus 1 - 1) 3 4) } \Downarrow 4$$

Question: could you write this proof..? What would happen if you tried...?

$$\frac{???}{(\text{if } (\text{plus } 1 \text{ } - 1) 3 4 \Downarrow 3)}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow n \quad n \neq 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

$$\frac{\quad}{(\text{if (plus 1 - 1) 3 4) } \Downarrow 3}$$

Answer: you **can't** write this proof,
because IfT will only let you evaluate
e1 when e0 is non-0!

$$\frac{???}{(\text{plus } (\text{plus } 0 \ 1) \ 2) \Downarrow 3}$$

$$\frac{???}{(\text{if } 1 \ (\text{div } 1 \ 1) \ 2) \Downarrow 1}$$

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1} \quad \mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

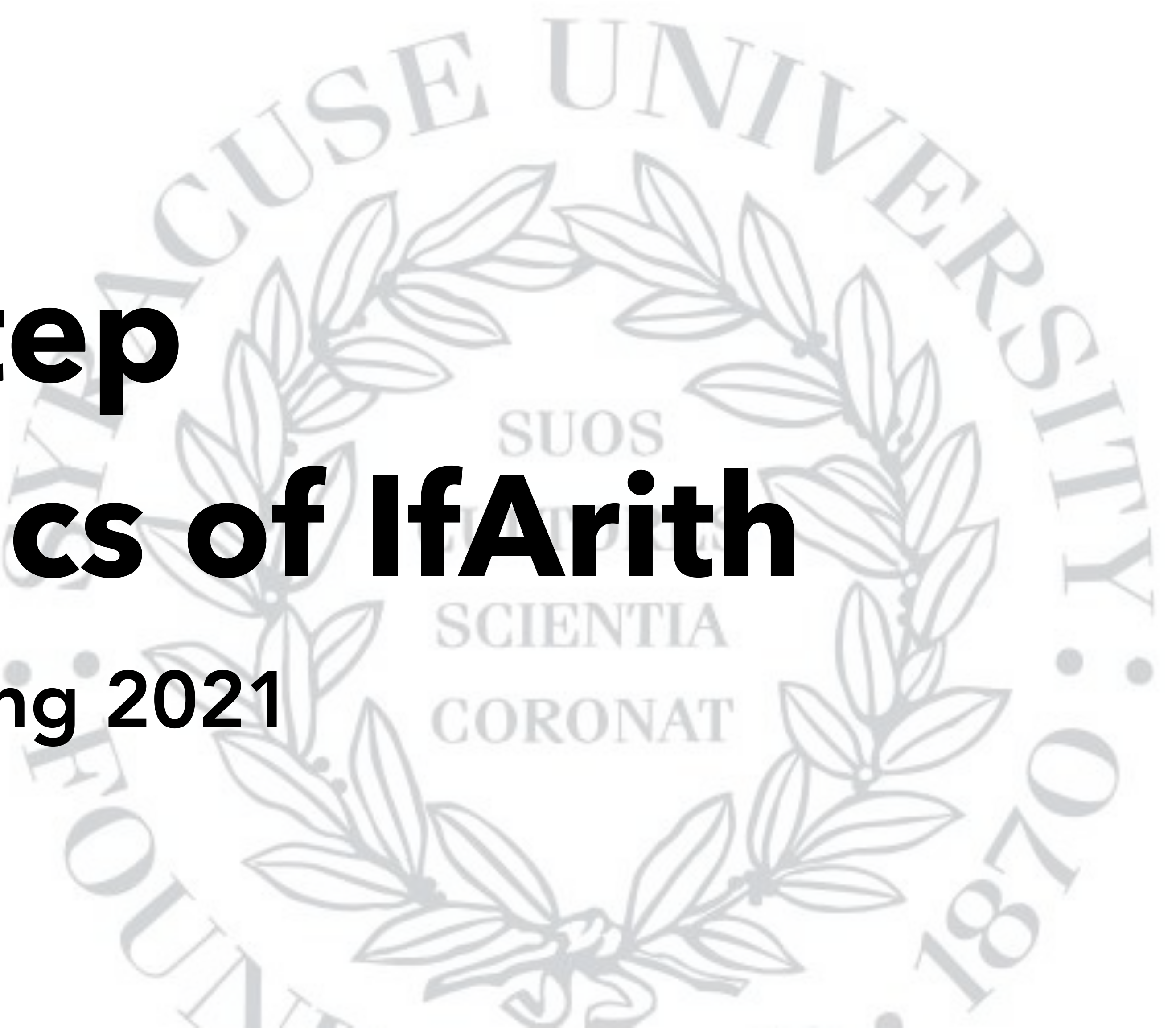
$$\mathbf{If}_T : \frac{e_0 \Downarrow n \quad n \neq 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

S

Small-Step Semantics of IfArith

CIS352 — Spring 2021

Kris Micinski



Code in the description!

Last Week: Defined **Big-Step** semantics for IfArith

Last Week: Defined **Big-Step** semantics for IfArith

Two different, but similar, formulations:

- Metacircular Interpreter in Racket
- Natural Deduction

The metacircular interpreter is our
“implementation” of natural deduction

```

(define (evaluate e)
  (match e
    [(? integer? n) n]
    [`(plus ,(? expr? e0) ,(? expr? e1))
     (+ (evaluate e0) (evaluate e1))]
    [`(div ,(? expr? e0) ,(? expr? e1))
     (/ (evaluate e0) (evaluate e1))]
    [`(not ,(? expr? e-guard))
     (if (= (evaluate e-guard) 0) 1 0)]
    [`(if ,(? expr? e0) ,(? expr? e1) ,(? expr? e2))
     (if (equal? 0 (evaluate e0)) (evaluate e2) (evaluate e1))]
    [_ "unexpected input"]))

```

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1} \quad \mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

```

(define (evaluate e)
  (match e
    [(? integer? n) n]
    [`(plus ,(? expr? e0) ,(? expr? e1))
     (+ (evaluate e0) (evaluate e1))]
    [`(div ,(? expr? e0) ,(? expr? e1))
     (/ (evaluate e0) (evaluate e1))]
    [`(not ,(? expr? e-guard))
     (if (= (evaluate e-guard) 0) 1 0)]
    [`(if ,(? expr? e0) ,(? expr? e1) ,(? expr? e2))
     (if (equal? 0 (evaluate e0)) (evaluate e2) (evaluate e1))]
    [_ "unexpected input"]))

```

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c} \quad \mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1} \quad \mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'} \quad \mathbf{If}_F : \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$


```

(define (evaluate e)
  (match e
    [(? integer? n) n]
    [(` (plus ,(? expr? e0) ,(? expr? e1))
     (+ (evaluate e0) (evaluate e1))]
    [(` (div ,(? expr? e0) ,(? expr? e1))
     (/ (evaluate e0) (evaluate e1))]
    [(` (not ,(? expr? e-guard))
     (if (= (evaluate e-guard) 0) 1 0)]
    [(` (if ,(? expr? e0) ,(? expr? e1) ,(? expr? e2))
     (if (equal? 0 (evaluate e0)) (evaluate e2) (evaluate e1))]
    [_ "unexpected input"]))

```

$$\mathbf{Const} : \frac{c \in \mathbb{Q}}{c \Downarrow c}$$

$$\mathbf{Plus} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0 + n_1}{(\text{plus } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Div} : \frac{e_0 \Downarrow n_0 \quad e_1 \Downarrow n_1 \quad n' = n_0/n_1}{(\text{div } e_0 \ e_1) \Downarrow n'}$$

$$\mathbf{Not}_0 : \frac{e \Downarrow 0}{(\text{not } e) \Downarrow 1}$$

$$\mathbf{Not}_1 : \frac{e \Downarrow n \quad n \neq 0}{(\text{not } e) \Downarrow 0}$$

$$\mathbf{If}_T : \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

$$\mathbf{If}_F : \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0 \ e_1 \ e_2) \Downarrow n'}$$

This week we'll be looking at **small-step** interpreters

Implement and formalize **textual reduction**

Small-step interpreters specify execution as a sequence of **steps**, where each step makes only a small, local computation

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

We will define the rules precisely in a few slides...

This allows us to reason about, and implement, control over execution in a fine-grained way at each step.

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

Allows us to reason about traces of the program more easily. Useful for things like...

- Reasoning about finite prefix of infinitely-looping programs (servers)
- Temporal properties of the program (data-race freedom, etc...)

Our job is to define this step function / operator,
written mathematically as $e_0 \rightarrow e_1$

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

First observation: can only take a step when both arguments to plus / div are **values**

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

We can immediately evaluate `(plus 2 2)` to 4,
and then to step the whole expression, we
substitute 4 in place of `(plus 2 2)`

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

We first identify a **redex** ("reducible
expression")

Now two rules (so far)

- Immediately reduce plus/div when args are values
- When e_0 or e_1 is **not** a value, reduce one of them and replace it

```
(div (plus 2 2) (plus 3 -1))  
→ (div 4 (plus 3 -1))  
→ (div 4 2)  
→ 2
```

- Immediately reduce plus/div when args are values

Let's translate this into the natural deduction style..

By the way, in this lecture we are defining a **new set of rules** for the small-step semantics, which I will call **SmallIfArith**

These rules are **separate** from the rules for **IfArith**

“Immediately reduce plus/div when args are values”

“Immediately reduce plus/div when args are values”

$$\mathbf{StepPlus} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0 + n_1}{(\text{plus } n_0 \ n_1) \rightarrow n'}$$

“When e_0 or e_1 is **not** a value, reduce one of them and replace it”

$$\mathbf{PlusLeft} \frac{e_0 \rightarrow e'}{(\text{plus } e_0 e_1) \rightarrow (\text{plus } e' e_1)}$$

$$\mathbf{PlusRight} \frac{e_1 \rightarrow e'}{(\text{plus } n e_1) \rightarrow (\text{plus } n e')}$$

The n here is a bit crucial: it adds determinism to our semantics!

“When e_0 or e_1 is **not** a value, reduce one of them and replace it”

$$\mathbf{StepPlus} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0 + n_1}{(\text{plus } n_0 \ n_1) \rightarrow n'}$$

$$\mathbf{PlusRight} \frac{n \in \mathbb{Q} \quad e_1 \rightarrow e'}{(\text{plus } n \ e_1) \rightarrow (\text{plus } n \ e')}$$

$$\mathbf{PlusLeft} \frac{e_0 \rightarrow e'}{(\text{plus } e_0 \ e_1) \rightarrow (\text{plus } e' \ e_1)}$$

“To process $(\text{plus } e_0 \ e_1)$, first check if e_0 is a value. If it is, then check if e_1 is a value. If both are, perform the addition.”

“When e_0 or e_1 is **not** a value, reduce one of them and replace it”

$$\text{StepPlus} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0 + n_1}{(\text{plus } n_0 \ n_1) \rightarrow n'}$$

$$\text{PlusRight} \frac{n \in \mathbb{Q} \quad e_1 \rightarrow e'}{(\text{plus } n \ e_1) \rightarrow (\text{plus } n \ e')}$$

$$\text{PlusLeft} \frac{e_0 \rightarrow e'}{(\text{plus } e_0 \ e_1) \rightarrow (\text{plus } e' \ e_1)}$$

These are the three cases you need to consider for +

Very similar operation for division...

$$\mathbf{StepDiv} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0/n_1}{(\text{div } n_0 \ n_1) \rightarrow n'}$$

$$\mathbf{DivRight} \frac{n \in \mathbb{Q} \quad e_1 \rightarrow e'}{(\text{div } n \ e_1) \rightarrow (\text{div } n \ e')}$$

$$\mathbf{DivLeft} \frac{e_0 \rightarrow e'}{(\text{div } e_0 \ e_1) \rightarrow (\text{div } e' \ e_1)}$$

$$\mathbf{PlusLeft} \frac{e_0 \rightarrow e'}{\text{(plus } e_0 \ e_1) \rightarrow \text{(plus } e' \ e_1)}$$

$$\mathbf{PlusRight} \frac{e_1 \rightarrow e'}{\text{(plus } e_0 \ e_1) \rightarrow \text{(plus } e_0 \ e')}$$

What would happen if we did this instead...?

Semantics would be **nondeterministic**

$((\text{plus } 1 \ 2) \ (\text{plus } 2 \ 2)) \rightarrow (\text{plus } (\text{plus } 1 \ 2) \ 4)$
 $((\text{plus } 1 \ 2) \ (\text{plus } 2 \ 2)) \rightarrow (\text{plus } 3 \ (\text{plus } 2 \ 2))$

$$\mathbf{PlusLeft} \frac{e_0 \rightarrow e'}{(plus\ e_0\ e_1) \rightarrow (plus\ e'\ e_1)}$$

$$\mathbf{PlusRight} \frac{e_1 \rightarrow e'}{(plus\ e_0\ e_1) \rightarrow (plus\ e_0\ e')}$$

This will manifest by complicating our definition of step

```
(define/contract (step e)
  (expr? -> expr?)
  ...)
```

We would need instead...

```
(define/contract (step e)
  (expr? -> (listof expr?))
  ...)
```

What about not..?

$$\mathbf{StepNot}_0 \frac{n \neq 0}{(\text{not } n) \rightarrow 0}$$

$$\mathbf{StepNot}_1 \frac{n = 0}{(\text{not } n) \rightarrow 1}$$

$$\mathbf{StepNot} \frac{e \rightarrow e'}{(\text{not } e) \rightarrow (\text{not } e')}$$

Finally, if...

$$\mathbf{If}_T \frac{n \neq 0}{(\text{if } n \ e_1 \ e_2) \rightarrow e_1}$$

$$\mathbf{If}_F \frac{n = 0}{(\text{if } n \ e_1 \ e_2) \rightarrow e_2}$$

$$\mathbf{If} \frac{e_0 \rightarrow e'}{(\text{if } e_0 \ e_1 \ e_2) \rightarrow (\text{if } e' \ e_1 \ e_2)}$$

So many rules! Rules are overly complicated: next lecture we will refactor them to be more attractive...

$$\mathbf{StepPlus} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0 + n_1}{(\text{plus } n_0 \ n_1) \rightarrow n'}$$

$$\mathbf{StepDiv} \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0/n_1}{(\text{div } n_0 \ n_1) \rightarrow n'}$$

$$\mathbf{PlusRight} \frac{n \in \mathbb{Q} \quad e_1 \rightarrow e'}{(\text{plus } n \ e_1) \rightarrow (\text{plus } n \ e')}$$

$$\mathbf{DivRight} \frac{n \in \mathbb{Q} \quad e_1 \rightarrow e'}{(\text{div } n \ e_1) \rightarrow (\text{div } n \ e')}$$

$$\mathbf{PlusLeft} \frac{e_0 \rightarrow e'}{(\text{plus } e_0 \ e_1) \rightarrow (\text{plus } e' \ e_1)}$$

$$\mathbf{DivLeft} \frac{e_0 \rightarrow e'}{(\text{div } e_0 \ e_1) \rightarrow (\text{div } e' \ e_1)}$$

$$\mathbf{StepNot}_0 \frac{n \neq 0}{(\text{not } n) \rightarrow 0}$$

$$\mathbf{If}_T \frac{n \neq 0}{(\text{if } n \ e_1 \ e_2) \rightarrow e_1} \quad \mathbf{If}_F \frac{n = 0}{(\text{if } n \ e_1 \ e_2) \rightarrow e_2}$$

$$\mathbf{StepNot}_1 \frac{n = 0}{(\text{not } n) \rightarrow 1}$$

$$\mathbf{StepNot} \frac{e \rightarrow e'}{(\text{not } e) \rightarrow (\text{not } e')}$$

$$\mathbf{If} \frac{e_0 \rightarrow e'}{(\text{if } e_0 \ e_1 \ e_2) \rightarrow (\text{if } e' \ e_1 \ e_2)}$$

One very important omission: there is **no defined step** for values!

These rules only tell us how to step expressions. We need to keep doing that (in a loop) until we reach a value.

Now that we have the rules, let's code them up as a small-step interpreter

```
(define/contract (step e)
  (-> (lambda (x) (and (expr? x) (not (value? x)))) expr?)
  'todo)
```