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"]))
```

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

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

$$\textbf{Not}_\mathbf{0}: \frac{e \Downarrow 0}{(\mathsf{not}\ e) \Downarrow 1} \qquad \textbf{Not}_\mathbf{1}: \frac{e \Downarrow n \quad n \neq 0}{(\mathsf{not}\ e) \Downarrow 0}$$

$$\textbf{If}_\mathbf{T}: \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\mathsf{if}\ e_0\ e_1\ e_2) \Downarrow n'} \qquad \textbf{If}_\mathbf{F}: \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\mathsf{if}\ e_0\ e_1\ e_2) \Downarrow n'}$$

```
(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"]))
```

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

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

$$\textbf{Not}_0: \frac{e \Downarrow 0}{(\mathsf{not}\ e) \Downarrow 1} \qquad \textbf{Not}_1: \frac{e \Downarrow n \quad n \neq 0}{(\mathsf{not}\ e) \Downarrow 0}$$

$$\textbf{If}_\mathbf{T}: \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\mathsf{if}\ e_0\ e_1\ e_2) \Downarrow n'} \qquad \textbf{If}_\mathbf{F}: \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\mathsf{if}\ e_0\ e_1\ e_2) \Downarrow n'}$$

```racket
(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"]))
```

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

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

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

$$\textbf{If}_\mathbf{T}: \frac{e_0 \Downarrow 0 \quad e_1 \Downarrow n'}{(\text{if } e_0\ e_1\ e_2) \Downarrow n'} \qquad \textbf{If}_\mathbf{F}: \frac{e_0 \Downarrow n \quad n = 0 \quad e_2 \Downarrow n'}{(\text{if } e_0\ e_1\ e_2) \Downarrow n'}$$

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"

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

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

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

$$\textbf{PlusRight}\ \frac{e_1 \to e'}{(\text{plus}\ n\ e_1) \to (\text{plus}\ n\ e')}$$

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"

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

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

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

"To process (plus e_0 e_1), first check if 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"

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

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

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

These are the three cases you need to consider for +

Very similar operation for division…

$$\textbf{StepDiv}\ \frac{n_0 \in \mathbb{Q} \quad n_1 \in \mathbb{Q} \quad n' = n_0/n_1}{(\mathsf{div}\ n_0\ n_1) \to n'}$$

$$\textbf{DivRight}\ \frac{n \in \mathbb{Q} \quad e_1 \to e'}{(\mathsf{div}\ n\ e_1) \to (\mathsf{div}\ n\ e')}$$

$$\textbf{DivLeft}\ \frac{e_0 \to e'}{(\mathsf{div}\ e_0\ e_1) \to (\mathsf{div}\ e'\ e_1)}$$

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

$$\textbf{PlusRight}\ \frac{e_1 \to e'}{(\text{plus } e_0\ e_1) \to (\text{plus } e_0\ e')}$$

What would happen if we did this instead…?

Semantics would be **nondeterministic**

```
((plus 1 2) (plus 2 2)) -> (plus (plus 1 2) 4)
((plus 1 2) (plus 2 2)) -> (plus 3 (plus 2 2))
```

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

$$\textbf{PlusRight}\ \frac{e_1 \to e'}{(\text{plus } e_0\ e_1) \to (\text{plus } e_0\ e')}$$

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..?

$$\textbf{StepNot}_\mathbf{0}\ \frac{n \neq 0}{(\textsf{not}\ n) \to 0}$$

$$\textbf{StepNot}_\mathbf{1}\ \frac{n = 0}{(\textsf{not}\ n) \to 1}$$

$$\textbf{StepNot}\ \frac{e \to e'}{(\textsf{not}\ e) \to (\textsf{not}\ e')}$$

Finally, if…

$$\textbf{If}_{\textbf{T}}\ \frac{n \neq 0}{(\mathsf{if}\ n\ e_1\ e_2) \to e_1}$$

$$\textbf{If}_{\textbf{F}}\ \frac{n = 0}{(\mathsf{if}\ n\ e_1\ e_2) \to e_2}$$

$$\textbf{If}\ \frac{e_0 \to e'}{(\mathsf{if}\ e_0\ e_1\ e_2) \to (\mathsf{if}\ e'\ e_1\ e_2)}$$

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

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

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

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

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

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

$$\textbf{DivLeft}\ \frac{e_0 \to e'}{(\text{div } e_0\ e_1) \to (\text{div } e'\ e_1)}$$

$$\textbf{StepNot}_0\ \frac{n \neq 0}{(\text{not } n) \to 0}$$

$$\textbf{StepNot}_1\ \frac{n = 0}{(\text{not } n) \to 1}$$

$$\textbf{StepNot}\ \frac{e \to e'}{(\text{not } e) \to (\text{not } e')}$$

$$\textbf{If}_\textbf{T}\ \frac{n \neq 0}{(\text{if } n\ e_1\ e_2) \to e_1} \qquad \textbf{If}_\textbf{F}\ \frac{n = 0}{(\text{if } n\ e_1\ e_2) \to e_2}$$

$$\textbf{If}\ \frac{e_0 \to e'}{(\text{if } e_0\ e_1\ e_2) \to (\text{if } e'\ e_1\ e_2)}$$

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)
```