Meeting 15: Operational Semantics, Type Checking

Homework 3
Time: 16.1 avg, 8.2 stddev, 15 median
Hard: 4.8 avg, 0.8 stddev, 5 median

What was hard?
- Thinking in terms of foldLeft/foldRight
- Dealing with Scala type errors
- add, isMagic
- induction proof

About You
- Starting to like functional programming
- Used to play varsity tennis
- Great, great grandfather helped found Telluride
- Play the bagpipes
- Favorite superhero is Batman
- Used to live in Poland
- Used to be a big baseball player
- Avid swing dancer
- Spent 3 weeks in Costa Rica teaching English
- Play broomball

Today: Questions (HW4?) Operational Semantics of Small, Type Checking Intro

Comments
- Really interesting and fun when the "lightbulb clicks"
- Starting to figure out Scala/becoming easier in each HW***
- Wasn't as difficult as HW2***
- Drills in recitation very helpful.
- Like homework and general questions at the beginning of class.
- Typing Boolean formulas for tests too tedious
- Wish for more concrete code examples
- Wish TAs help with getting Scala syntax right -- no problem
- Conceptually easy but code takes time.

Ask Questions. Our goal is to guide you through the learning in the homeworks.
Questions

4. RegExp
   Boolean operators - Yes

5. When are these type errors

```python
def matchReDo(re: RegExp, chars: List[Char], sc: List[Char] => Boolean):
    ("abc" = "" + "abc") Boolean =
    "If a prefix of `chars` matches the reg
    expr `re`, then sc is called on the
    suffix/remainder

    def matchReDo(re, chars) match {
        case (ab: String, _) => false
        case (EmptyString, _) => sc(chars)
        case ((concat(r1, r2), _) =>
            matchReDo(r1, chars, _r) matchReDo(r2, _r, sc)
        case (Star(r), _) =>
            sc(chars) || ...
                try 1
                try 2
                try 3
```

"ab"
Operational Semantics

- 1. What does an operator do?

+ 

- 2. Evaluation order

\[ e_1 + e_2 \]

Evaluation Judgement

\[ e \rightarrow e' \]

Expression \( e \) takes one step of reduction to an expression \( e' \).

Define a judgment using inference rules
def match_re (re: RegExp, s: String)
    match re do
        rem match &
        case Nil => true
        case _ => false
    end

\[ e \rightarrow e' \]

\[ e'' = l_e_1 + e_2 \ln \]

\[ \text{DefPlus} \]

\[ n' = n_1 + n_2 \]

\[ \frac{n_1 + n_2}{n_1 + n_2} \rightarrow n' \]

\[ \frac{\text{Plus}(n_1, n_2)}{n_1 + n_2} \rightarrow n' \]

so far this is my definition for \( e \rightarrow e' \)
\[(1+2)+3\]

\[\text{Step Plus Left}\]
\[e_1 \rightarrow e_1' \]
\[e_1 + e_2 \rightarrow e_1' + e_2\]

\[1 + (2+3)\]

\[\text{Step Plus Right}\]
\[e_2 \rightarrow e_2' \]
\[n_1 + e_2 \rightarrow n_1 + e_2'\]

---

\[\frac{3 = 1 + 2}{\text{do plus}}\]
\[\frac{1 + 2 \rightarrow 3}{\text{step plus left}}\]
\[(1+2)+3 \rightarrow 3 + 3\]

\[\text{enforces left to right evaluation of +}\]

---

\[\text{Bill's Alternative Step Plus Right}\]
\[e_2 \rightarrow e_2' \]
\[e_1 + e_2 \rightarrow e_1 + e_2'\]

---

\[\text{along with Step Plus Left}\]
\[\uparrow \text{non-deterministic eval of +}\]
1. \((1+2) + (3+4) \rightarrow 3 + (3+4)\)

and

2. \((1+2) + (3+4) \rightarrow (1+2) + 7\)

with Bill's rules

the 4/25

def step(e: Expr) : Expr =

true + 3 — this expression is stuck
3(5) → type error