Project Proposal - Types Languages Compilers

Project Name: **JustYourRegularBRzOs**
Team Members: Arman Jindal asj2152, Jonathan Cappell jec2274

Goal: Take in a string, convert it to a 'regex', and construct a DFA (recognizer) using Brzozowski regular expression derivatives.

Method: Implement the mathematical formalism described in Owens, Reppy, and Turon *Regular-expression derivatives reexamined* (2009)
We have defined discrete steps below to be taken to implement this Goal.
Task:

1. Define a Regular Expression (RegExp) data type and Parser

Recursively define a RegExp data type in the Haskell constructor and a checker function for it.

$$
\begin{array}{rcll}
r, s & ::= & \emptyset & \text{empty set} \\
 & | & \varepsilon & \text{empty string} \\
 & | & a & a \in \Sigma \\
 & | & r \cdot s & \text{concatenation} \\
 & | & r^* & \text{Kleene-closure} \\
 & | & r + s & \text{logical or (alternation)} \\
 & | & r \,\&\, s & \text{logical and} \\
 & | & \neg r & \text{complement}
\end{array}
$$

2. Create the helper function v(r) that checks if r is nullable.

$$\nu(r) = \begin{cases} \varepsilon & \text{if } r \text{ is nullable} \\ \emptyset & \text{otherwise.} \end{cases}$$

and is defined as follows:

$$
\begin{aligned}
\nu(\varepsilon) &= \varepsilon \\
\nu(a) &= \emptyset \\
\nu(\emptyset) &= \emptyset \\
\nu(r \cdot s) &= \nu(r) \,\&\, \nu(s) \\
\nu(r + s) &= \nu(r) + \nu(s) \\
\nu(r^*) &= \varepsilon \\
\nu(r \,\&\, s) &= \nu(r) \,\&\, \nu(s) \\
\nu(\neg r) &= \begin{cases} \varepsilon & \text{if } \nu(r) = \emptyset \\ \emptyset & \text{if } \nu(r) = \varepsilon \end{cases}
\end{aligned}
$$

3. Compute the derivative of a, with respect to a symbol **a**:

$$
\begin{aligned}
\partial_a \varepsilon &= \emptyset \\
\partial_a a &= \varepsilon \\
\partial_a b &= \emptyset \quad \text{for } b \neq a \\
\partial_a \emptyset &= \emptyset \\
\partial_a (r \cdot s) &= \partial_a r \cdot s + \nu(r) \cdot \partial_a s \\
\partial_a (r^*) &= \partial_a r \cdot r^* \\
\partial_a (r + s) &= \partial_a r + \partial_a s \\
\partial_a (r \,\&\, s) &= \partial_a r \,\&\, \partial_a s \\
\partial_a (\neg r) &= \neg(\partial_a r)
\end{aligned}
$$

The rules are extended to strings as follows:

$$
\begin{aligned}
\partial_\varepsilon r &= r \\
\partial_{ua} r &= \partial_a (\partial_u r)
\end{aligned}
$$

4. Create a Haskell data type of a DFA
5. Use the v(r) function and the RegExp derivatives to create a DFA implementing the algorithm below:

```
fun goto q (c, (Q, δ)) =
    let q_c = ∂_c q
    in
      if ∃q' ∈ Q such that q' ≡ q_c
        then (Q, δ ∪ {(q, c) ↦ q'})
        else
          let Q' = Q ∪ {q_c}
          let δ' = δ ∪ {(q, c) ↦ q_c}
          in explore (Q', δ', q_c)

and explore (Q, δ, q) = fold (goto q) (Q, δ) Σ

fun mkDFA r =
    let q_0 = ∂_ε r
    let (Q, δ) = explore ({q_0}, {}, q_0)
    let F = {q | q ∈ Q and ν(q) = ε}
    in ⟨Q, q_0, F, δ⟩
```

Fig. 1. DFA construction using RE derivatives

6. Include test cases with RegExps and strings that are in and out of their language. Import the external RegularExpression checker and validate it against the generated DFA to test that we have correctly implemented the algorithm.

All images here are taken from the paper *Regular-expression Derivatives reexamined* By Scott Owens, John Reppy, and Aaron Turon.

Questions:

- How much external code can we use/look at/import, especially for Regex and DFA things?
- Can we use an external library for building and running a DFA. i.e https://github.com/AlexanderJDupree/DFA-Check or should we build our own?
- Can we use an external library for testing our generated DFA against a working (external) RegExp evaluator, i.e https://github.com/cacay/regexp
- Can we use an external library for building and running a Regex. i.e https://github.com/cacay/regexp or should we build our own?

This project is based on John Hui's helpful recommendation for projects on EdStem, here: https://edstem.org/us/courses/36106/discussion/2893941.

References:

https://www.ccs.neu.edu/home/turon/re-deriv.pdf
https://dl.acm.org/doi/pdf/10.1145/321239.321249