Journal Report 6

10/7/19-10/11/19
Rahul Mittal
Computer Systems Research Lab
Period 2, White

Daily Log

Monday October 7

Sick

Tuesday October 8

Spent half an hour doing a math problem with Bryan. Started writing the Haskell thing, made a Formula datatype with cases Constant, Sum, Product, Combination, Factorial, added operators for adding and multiplying formulae, added operator for combinations, added short 'fact' function for factorials so that 'Factorial' isn't necessary, added Difference and Quotient cases of the Formula datatype, added Variable case of the Formula datatype. Start writing function to convert Formulas into JSON strings, start with Constants, that was easy, then do Variables, also easy, go to Sums, write helper function 'jsonObject' for creating JSONObject strings with less typing, create 'jsonArray' helper function for converting list of JSON strings to JSONArray string, finish json function for rest of Formula cases. Try testing a sample formula against the Haskell, realize I have to make too many changes to the formula to test it, in main program, write 'prepareFormula' function to slightly convert raw formula string into using necessary operators to be parsed with my Haskell thing, put sample formula through 'prepareFormula', fix bugs, put prepared output into Haskell, works. Look up how to do command line stuff from Java, find ProcessBuilder thing for next time.

Thursday October 10

Start writing thing to compile and run Haskell program from main program, write code that creates a new ghci process (Haskell interactive environment), writes in all the lines of my Haskell code, then evaluates the prepared formula text. This doesn't work, I don't exactly know why, the imports are being weird. Change to compiling my Haskell code first by doing 'ghci formulae.hs' then evaluating prepared formula text, works.

Making up lost time from Monday

Add subclasses of Quantity to represent formulae, write 'jsonToQuantity' function to convert JSON into instances of those subclasses, done.

Timeline

Date	Goal	Met
September 30	Revise function identifying actions and continue process to the point that 95% of random proofs on the Internet produce the correct output	No, only finished Colorado proofs
October 7	Modify function identifying actions to include information about statements made, meaning what verb used, what nouns acting/being acted on	No, but layed foundation for doing it
October 14	Create Haskell DSL for converting formulae into JSON and be able to compile/run Haskell program from my main program	Yes, also have formulae as instances of Quantity
October 21	Get detailing function to work with two templates	
October 28	Get detailing function to work on sample proof + first two Colorado proofs	

Reflection

This week was straightforward, just like I thought it would be. The idea of creating a DSL in Haskell for the formulae worked well enough. It still feels like overkill, especially considering that the 'prepareFormula' function actually has to make a lot of changes, but it didn't take that long compared to what writing my own parser or learning to use some other actual parser would have taken, so I guess it works.

Here's the sample formula I tested:

```
(fact u / (fact b * fact (u - b))) * fact b
```

Here's the prepared version:

```
(fact (Variable 'u') `Quotient` (fact (Variable 'b') *** fact ((Variable 'u') `Diff
```

Here's the JSON output:

```
{
  "type": "product",
  "mands": [
    {
      "type": "quotient",
      "top": {
        "type": "factorial",
        "of": "u"
      },
      "bottom": {
        "type": "product",
        "mands": [
          {
            "type": "factorial",
            "of": "b"
          },
          {
            "type": "factorial",
            "of": {
              "type": "difference",
              "left": "u",
              "right": "b"
            }
          }
        ]
      }
    },
    {
      "type": "factorial",
      "of": "b"
    }
  ]
}
```

And here's the final form of the formula as an instance of Quantity:

```
Product(
    qs=[
        Quotient(
            top=Factorial(q=Variable(name=u)),
            bottom=Product(
                qs=[
                    Factorial(q=Variable(name=b)),
                    Factorial(q=Difference(
                        left=Variable(name=u),
                        right=Variable(name=b)))
                ]
            )
        ),
        Factorial(q=Variable(name=b))
    ]
)
```