Journal Report 7
10/15/19-10/25/19
Rahul Mittal
Computer Systems Research Lab
Period 2, White

Daily Log

Tuesday October 15

Not in school

Thursday October 17

Start writing code to match trees, make matchIntoParameters(node, tree) helper function, make function into 'when' statement, contemplate not using a helper function, decide against not using helper function, add another nested 'when' statement for when the node is a TemplateParameter, using TemplateParameters.copy() function to construct return values for those cases, now doing other main cases, added operator function for adding two TemplateParameters instances, finished helper function, now adding 'findStatement', 'findQuantity', 'findMathObject' functions, added checking through all templates to each one, added if statement for checking if it's a formula in 'findQuantity', realize I need something like that in 'findStatement' as well.

Monday October 21

Finish inside of said if statement, do the same whole thing for 'findStatement', remember that I already have an isPlaceholder function, replace my current almost identical if statement with a call to isPlaceholder, finished that thing in 'findStatement', added case for arbitrary math objects ('ball', 'urn', etc.) to 'findMathObject', replace 'TODO()' with 'return null' in all of the 'find' functions.

Tuesday October 22

Add new 'getStatements' function that goes through the tree and calls 'findStatement' to get statements, add 'StatementNode' class to be return in a list by 'getStatements', just call 'findStatement' on each node, if it doesn't work, call it on all children, start testing new identification stuff, add new main function that calls 'getStatements', run against that sentence, doesn't work, print out all 'matchIntoParameters' calls, do a lot of debugging, it works! yay, start working on next part of the sentence, try inputting simplified form to see constituency tree, get NoClassDefFoundError about some Dijkstra thing, google it, google it more, decide to just add a compiled dependency to the project and rerun it, works, realize that it was probably because I had rebuilt the project before inputting the next sentence, changing the jarfile.

Thursday October 24

Look at constituency tree for next sentence, it's really annoying, think for a while, decide to try other approach, make 'BetterParser' file, created 'SyntaxElement' sealed class with two subclasses: Token and SyntaxParameter, start trying to write parsing function, add initial overall

function, decide to use 'parseHelp' helper function like I did before, add 'ParsePosition' class to use as states inside of 'parseHelp' function traversed using DFS/BFS, realize that parsing Statements/Quantities/MathObjects inside of parsing other ST/QQ/XN will become not nice if I write it like this, change to using BetterParser class to parse things, BetterParser contains stack variable and has 'tryParse' function to return ParseResult¡S¿ instances when called, so if something doesn't work the next thing can be returned, add initialization statement for BetterParser.

Making up lost time from Tuesday

Copy over statements for declaring 'STATEMENT_TEMPLATES', 'QUANTITY_TEMPLATES', 'MATH_OBJECT_T
change 'Template' to 'SyntaxTemplate', starting writing DP approach to parsing, use dp[x][y][type]
as parsed thing of type 'type' between x and y with the minimum amount of words skipped.

Timeline

Date	Goal	Met
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	Yes, one statement template and one math object template.
October 28	Get detailing function to work on sample proof + first two Colorado proofs	No, haven't finished sample proof.
November 6	Get simplified parser to work on sample proof + first two Colorado proofs	
Novemeber 11	Make sure simplified parser works on all Colorado proofs	

Reflection

This week was really annoying. I was able to get the function matching trees itself to work, and tested it against that part of that sentence in the sample proof.

The part of the sentence was: We can choose b urns, which will contain a ball, in c ways.

The result StatementNode was:

```
Equals(
    q1=Amount(
        obj=Choice(
            noun=ArbitraryMathObject(noun=(NP (DT a) (NN ball)))
        )
    ),
    q2=Formula(
        placeholder=(NN g)
    )
): (ROOT (S (NP (PRP We)) (VP (MD can) (VP (VB choose) (NP (NP (DT a) (NN ball)) (P
```

However, I then went on to the next part of that sentence in the sample proof, and (after a bit of stupidity with a NoClassDefFoundError) looked at its constituency tree and realized that it was too complicated. It wasn't too complicated to reproduce as a template if my project dependended on just that, but this is something I would have to do for a large variety of general combinatorial/mathematical statements, which is essentially infeasible and ultimately probably harder than just manually transcribing the proofs as an instance of my Statement class.

A more ideal situation would be one where the part of the template saying what the input English text should look like could literally just itself be normal English text. I initially intended to do this with the matching trees approach, but decided against it because of how similar sentence structures can result in very different trees. Unfortunately, that also makes the matching trees approach difficult in general.

My new, simplified approach is to write a parser where I can express the templates as something like "we can choose %xn1% in %qq1% different ways", and then the parser will just check

if the input text is of that form, where '%xn1%' has to be filled in with an actual math object (and similarly for '%qq1́). This approach will have a much greater guarantee of success because it depends only on the input text itself as a list of tokens and not on whatever kind of tree Stanford wants to make. The drawback will be that the input will have to be more constrained than I was hoping for with the matching trees approach (but that I didn't actually know how I could achieve with the trees). This will be mitigated a little bit by allowing the input text to have unused words that aren't matched to any part of any template, but it's still a large compromise.