Journal Report 12

12/9/19-12/13/19
Rahul Mittal
Computer Systems Research Lab
Period 2, White

Daily Log

Monday December 9

Add Journal 11 to my project files, commit to GitHub, start work on presentable main function.

Tuesday December 10

Move contents of previous main function into while loop in new main function, check for sentence not being empty string, loop through list of output, refactor code for printing data classes nicely into dataStringWithNewlines function, add if statement for printing output, finished main function for now, go back to Colorado proof 2, add LogicalStatement subclass of Statement, add Since subclass of LogicalStatement, add comment in section for statement templates separating logical statement templates from others, add "since x y" statement template.

Tuesday December 12

Start transcribing the proofs, finished proof 1, changed name of LogicalStatement to MetaStatement, allow formulae to be math objects as well (but only if they're plain variables), add DefineMathObject and DefineQuantity subclasses of Intent, brain fart, change signature of those constructors (that's a bad way to say it), add ObjectVariable subclass of MathObject, finish the code in the dfs to use ObjectVariable, replace all outside usages of ParsedBlah constructors with parsed function, add "let xn1 be xn2" metastatement template, now going to make output into a file as well, finished conversion to using StringBuilder, almost forgot to add an empty newline in between things, Googled "kotlin write to file", wrote thing using printwriter but then replaced with Kotlin iDiOmAtIc WaY, going to test now on proof 1, created output folder, fixed minor error in proof 1, getting empty sentence on something which should not be empty sentence, make it so that in the output the sentence has been trimmed so that it doesn't look stupid, add optional "[hand]" to left/right side template, add 'this' option for HangingAmount, also create input directory for better organization, move colorado_(1—2)_final.txt to input directory and rename to colorado_(1—2).txt, remove parenthesis left stranded inside of string, rerun, delete file colorado_1_out.txt, now transcribing proof 2, fix typo in "there are" template, add 'element' to math object words, add subset to math object words, oh subset was already there, obscure bug, transcribe proof 3, add And subclass of Statement, add "both sides of the equation count" statement template, weird bug showing up again, modify newlining method to not have the extra spaces after the commas.

Timeline

Date	Goal	Met
November 25	Integrate actual formulae (as opposed to placeholders) into output as Quantity objects, make sure parser works on first 2 Colorado proofs	Yes, but need more functionality to better represent Colorado proof 2
December 9	Make sure parser works on rest of Colorado proofs, parse statements of intent, add optional functionality to syntax, add thing in syntax for "does not"	Mostly - proof 4 unfinished, proof 5 unfinished but straightforward
December 6	Finish Colorado proofs, prepare code to accept whole proof as input for demonstration	No, Colorado proofs unfinished, yes, code prepared
Winter Goal	Take in as input a combinatorial proof, with no modifications other than expressing formulae with my formula syntax, and output the claim and all steps of the proof as instances of Statement.	See below

Reflection

So this week was disappointing. I'll start with what I did fully accomplish - the preparation of my code for a demonstration to others.

This was fairly straightforward, as it should have been. What I decided on was to have the user write their proof in a plaintext file, then prompt the user for the name of that input plaintext file, as well as the name of the file where the output would go (the amount of output made it so that just printing the output would be too unwieldy, especially considering the debugging information being printed that would be annoying to remove and might be nice to keep). My program would find all of the sentences in the proof, run parseStatementOrIntent on each one, and then output, for each sentence, the original sentence, and either "Empty sentence" if the parser found nothing at all, or the amount of unused words and then the parsed statement/intent.

Then came actually testing this demonstrable code on the Colorado proofs. I transcribed Colorado proof 1, then tested it on my program. After fixing a few things in the proof and a few things in my code it worked, though it did remind me that if the input proof has mistakes concerning things like the formulae using incorrect syntax or single letters other than those that represent actual words being present, since those should instead be formulae, the program will throw an exception somewhere and it will frequently be really difficult to tell what the problem is, which isn't ideal.

Moving to proof 2, the process was similar, but it seemed like there was quite a bit that I had to fix to make it fully work considering that I had previously moved to (and then finished) proof 3, and so should have pretty much been done with getting proof 2 to work. Proof 3 went pretty much the same way as proof 2 so I won't comment on it separately.

One thing which is worrying, though, is that there's this weird bug I found (in both proofs 2 and 3), where a simple statement that should have given a proper output instead gave "Empty

sentence." In both cases, the statement(s) where this happened were very similar to another statement in the proof that worked - in proof 2, there were 2 similar statements, and the first one didn't work; in proof 3, there were 4 similar statements, and the second and third ones didn't work. The really weird thing is that even when I made the sentences exactly identical, those same ones still didn't work while the ones that worked before still did. Clearly, something really weird is going on here - maybe something to do with certain characters not being the characters that I think they are or something - and it's going to be really annoying to fix, which is not good.

The most disappointing part of this week, however, was probably just the fact that I didn't even get to proofs 4 and 5. In thinking about it some other time when I wasn't working on the project, I thought of a way that I might reasonably parse that problematic sentence in proof 4, which is below:

When creating a subset of S, for each element of S, there are two options: to include it or not to include it.

I was thinking about creating a template like "when creating %xn1%, for each %xn2%, there are %qq1% options" or something similar, and it would be represented as something like Equals(Amount(it.xn(1)), Amount(ForEach(something))) (I would have to add some kind of new classes to represent choosing something for multiple objects, but it would be more generally applicable and so wouldn't be weird to add). It's obviously a template very much tailored to that sentence in proof 4, but if we need proof 4 to be parsed, then it's necessary and should work.

However, with the time that it took to make proofs 1 through 3 work with the demonstrable code, I didn't even get to trying to do that, which is disappointing. It's even more annoying that I couldn't even get proofs 2 and 3 to fully work because of that bug that I have no idea of the cause of.

Ultimately, on my winter goal, you can look at it in a few ways. If my goal was to accept *any* combinatorial proof, that's clearly impossible with my approach, because you could just look at all of my templates and write a proof that didn't fit any of them. The more reasonable ultimate goal would be to be able to accept most proofs that would reasonably be written, and I can't even be sure that that goal is possible with my approach, but that's the risk that I took.

The more disappointing thing is that the benchmark I set in my mind of having all of the Colorado proofs ultimately didn't materialize. I fully believe that with more time spent on the project I would have achieved that, so in that respect a lack of diligence of mine is partly to blame. There is also this week left to fix that bug and finish proofs 4 and 5, so it's not over as far as meeting that benchmark.

I think I can at least say that I've demonstrated that my parser itself does indeed work fully and at the very least that for a given combinatorial proof, you can modify the templates - which requires almost no actual coding - to be able to parse your proof, which is at least something, if not what I wanted to have achieved by now.