Journal Report 17

2/10/20-2/14/20
Rahul Mittal
Computer Systems Research Lab
Period 2, White

Daily Log

Monday February 10

Go back to Colorado 1, implication from statement 5 to statement 6 covered by adjacent implications, now need implication from 4 to 6, 4 and 6 would have common subsegments if desired quantity was replaced with actual value, create function replaceHangings: Statement -¿ List¡String¿ inside getStatementsAndImplications function, create dfs function on ParsedThings inside replaceHangings function, create context function (similar to parallel function) for convenience, perform dfs (on all Things) inside of dfs to extract contained ParsedThings.

Tuesday February 11

Do some autocomplete thing so that "reflection is in the classpath" or something like that, add PARSED_CLASSES list of ParsedThing classes, dfs in dfs done, for each index inside focus of main dfs, add corresponding token to result if not contained in any sub-ParsedThing, otherwise dfs on that sub-ParsedThing, make replaceHangings its own function outside of other stuff for easier testing, now I need to actually do the replacing of hangings, add check in beginning of dfs function to replace if the argument is of type HangingAmount, remember that the to property of ParseContext is exclusive and remove the +1, now, loop through all statements, for all statements that currently imply that statement, if statements contains "this", treat as reference to previous statement, declare IMPLICATION_SIMILARITY_THRESHOLD of 5 (tokens), get replaced version of original statement, compile set of all statements sharing a substring of at least the threshold length with the replaced version, change implications list to a set for convenience, now, for all statements sharing sufficient length substring and being referenced by a following statement that already implies the original statement, add implication to original statement, go back to desired quantity loop to also extract text of desired quantity, modify parallel function to act as identity when argument isn't a ParsedThing, create unwrap function similar to parallel function that maintains inner parsed context information, get tired of having to keep track of what tokenization everything comes from, add the actual text to ParseContext, make necessary modifications in parser, now just use normal wrapped version of quantity as desired quantity so no need for additional text variable, make necessary modifications in desired quantity finding to use unwrap instead of parallel.

Thursday February 13

Now handling Colorado 3, for statements "by" multiplication principle, take simplistic approach, find last two Equals statements with q1 or q2 being Amount, add implication from those two statements if both exist.

Timeline

Date	Goal	Met
January 20	Finish conversion of sentences to proper statements, finish adjacent implications from "Therefore", find similar phrases used in different statements	Yes, yes, yes but haven't used them for anything yet
February 3	Be able to find all implications between statements in Colorado proofs	No, worked on figuring out hanging quantity
February 17	Be able to find all implications between statements in Colorado proofs, start work on combinatorics foundation in Coq	Finished implications for Colorado 1, 2, 3, not started Coq
February 24	Start work on combinatorics foundation in Coq	
March 2	Continue work on combinatorics foundation in Coq	

Reflection

This week was okay; I was able to finish the code to find all implications in Colorado proofs 1 through 3, but once again had much less time than I anticipated because presentations took much longer than I expected and so didn't have enough time to start on Coq, which I now urgently need to start this Tuesday.

I spent most of my time on handling the case of Colorado 1. Finishing this case was satisfying in that to some extent it validated essentially all of the work I've been doing on finding implications: one of the implications was a simple adjacent implication, which I've already handled, and for the other, the two statements would share an important substring (to choose a subset of size k) that I could find with my code for finding similar substrings, except that the implication statement used desired quantity instead, so I just had to write code that replaced the desired quantity placeholder text with the text describing what the desired quantity actually was, and then compare this modified statement text in order to properly find the implication.

As much as it was a validation, however, it was also somewhat not a validation, in that there were a lot of modifications to previous code that I had to make in order to get the information that I actually needed rather than the information that I thought I needed. The most recent example is the desired quantity - the code that I wrote last week found the actual desired quantity Quantity object, but with the approach I took above, I needed the text that generated that Quantity. Unfortunately, since I took the approach of using the parallel function to simplify my code for finding the desired quantity, the result didn't have any information about where it came from in the text, so essentially, what I thought was saving me time and making my code simpler was just a waste of time. (I did use parallel somewhere else in new code this week so it wasn't a complete waste, but my original use of it was still useless).

Probably the most annoying "non-validation" was the whole thing with having to keep track of which tokenized sentence each statement came from. I've been doing this using a SentenceTag, meaning that every time I wanted to get the text that a statement came from, I need to look up that SentenceTag in my list of tags, and then access the tokenizations list, which also means that I need to pass that list to any functions that use the text. This was okay when it was in a single place, but I ended up needing the original text a lot this week, so it basically became a huge annoyance, and at the end, when I needed the text of the desired quantity, I realized that even if I now had the ParseContext from not using parallel anymore, I still needed to know which sentence it came from, but I find desired quantity from intents, and I hadn't been keeping track of which sentence each intent came from, so I just ended up adding the actual text to the ParseContext because it was an easy modification to make in the parser, meaning that all of the work I had already done in other parts of the code to get the original text could now be done much more simply, but that code was already written so I was now stuck with code that took longer to write and was messier while simultaneously having an easier way to do it. That was all very annoying, but at least it's over now, I guess.

I was going to write something here, but I realized that it would essentially be a rehash of the last paragraph from last week's journal, so basically: I really need to do this Coq stuff now.