Theorem Prover

Reuben Tanner

Intro

When first sitting down to tackle this program I was unsure of a lot of things and, to make matters worse, I had to miss the lecture where you went over all the different examples of how to resolve sentences together. Since this domain is pretty foreign to someone who hasn't had much experience with first order logic it took quite a bit of work to get the background of how to solve these problems but once that happened, the magic began to roll. When thinking about how to design this the objects required for this program, I decided that a functional approach would suit the problem well so, for the most part, my sentences, predicates and KB are all immutable.

First phase

For the first phase of this project, I spent a lot of time reading chapter 9 and googling articles and trying to actually figure out how to unify two predicates and how to resolve two sentences. The most helpful thing throughout this was looking at the section in the book where we proved that Colonel West is a criminal. Once I began to understand what was going on and had completed a parser that complied with the grammar, I did a stupid thing and tried to do waaaaaay too much of the resolution stuff all at once (noobie mistake). After floundering for a couple hours I stopped myself and threw away all the code besides the parser and began anew using TDD.

Second phase

After I started to actually code correctly, things started falling into place *much* better and the assignment became far more fun. I copied the unify algorithm nearly verbatim which really helped me not have to think about the complexities that were involved with unification. Of course, after I had the algorithm working it was quite clear that unifying two predicates was not something that was very difficult. Unit tests worked wonderfully for the nature of this program because I would simply write two sentences that should or should not resolve and then the result and make sure they were equal. I followed a similar approach with the sentences.

Third Phase

The third and final phase of my development involved me attempting to come up with a clever way to perform resolution on the KB. This perplexed me at first because I wasn't sure if I should be removing statements from the KB after resolving them since I technically already used them or what the proper procedure was. I tried to do a recursive function that would recurse upon itself after resolving a sentence and removing the two precursor sentences from the KB. This didn't work out so hot for anything beyond the file test2. I was in the process of refining the recursive routine when you sent out the email saying you had extended it and at that point I called it quits for a little while. After taking some time away from the program and coming back to it and after reviewing the output from the random resolver it became clear to me that we aren't supposed to be removing anything from the KB and we don't have to use all the sentences in the KB. If we can simply find two sentences that are either derived from the KB (like the set of support) or presently in the KB that resolve to nothing then we have found a solution. Once this breakthrough occurred, I was able to complete the solver.

Heuristics

What I did to speed my program up and reduce the number of resolutions and time required was implement a combination of heuristics mentioned at the end of chapter 9. I first attempted to do subsumption but got a little overwhelmed because it wasn't very clear what exactly made a non-trivial statement more specific than another non-trivial statement so I fiddled with it for a little bit and gave up. I then decided to implement unit preference by providing each sentence with a weight based on how many predicates and variables it contained. I used this weight scheme in the compareTo function for sentences so that I could sort lists in such a way that the unit clauses would be first when doing resolution. I also implemented a set of support by storing all sentences derived from the resolutions performed on the refuted goal and using the set in every subsequent resolution. These two together perform ridiculously fast resolutions; test5.3 takes ~100ms to complete!

Conclusion

Overall, I thought this problem was actually pretty hard but once I started understanding exactly how resolutions and unifications with first order logic statements worked and remembered efficient methodologies for software development things started to look a little more hopeful.