CS 830 Intro to AI, Spring 2025

Written Assignments, Sushma Akoju

Assignment 7

1. Describe any implementation choices you made that you felt were important. Clearly explain any aspects of your program that aren't working. Mention anything else that we should know when evaluating your work.

    a) I wrote my implementations in parts, by scaffolding - first I wanted to write my own parser. It worked and as I focussed more on details. By Saturday night I had the parser working with my unify method (is the part for assignment6). I also ran a lot of test cases for each one of my parts of implementations as any small error in the parsing can block everything that comes later in the algorithm (as assignment 7 datasets were nested and intentionally made complex). Actually all of the tests individually on each one of my functions passed successfully not only on parser, but also on unify, substitute, occur_check, and resolve.

    b) For the parser, I used an iterative parser (to avoid infinite loop/max recursion) using a stack. After implementing the entire parser, I found some (very few) examples requiring function names and even predicates being alphanumeric, which is not what I initially used for **function names and predicates**.

    c) My parser works faster. Once I parsed all clauses, I extracted function names, predicate names, constants and variables for unify algorithms and for resolution implementations.

    d) Smaller clauses: I selected the smaller clauses as if I was debating whether it has to be the number of literals or the number of nested parentheses. While parsing, I have collected the depth of each clause and maintained clauses_length list to sort it in ascending order for later.

e) Unification and substitution do work fine, it was not difficult (it is nearly the same as in assignment 6) but I included occur_check as required for assignment 7.

f) But I am seeing sometimes unify returns None, and the history update happens but it keeps on returning None. This was the most time consuming task. To debug this - it took me all Monday morning - this evening at around 7pm. The And I had to debugged this (without sleep) non stop, 3 days, to find out: I was calling unify():

```
if isinstance(exp1, str):
        if isinstance(exp1, str):
```

Exp1 in the second line should have been exp2.

g) The Resolve method is also fine.

h) I was mainly working on a couple of approaches around how to store the history and discard it - which was the most crucial part. For the most part I thought I was doing something wrong with history updates i.e. each time I substitute something, the history needs to be updated so that the algorithm can discard and restart. Using memoization. It was taking longer only because I wanted to achieve some non-trivial way to stop the prover to make it realize it cannot solve it.

i) If I run on all possible pairs of clauses in a while loop and smaller clauses are already resolved (not in history), then it reaches max recursion depth for unify.

j) Unify needs to be implemented using an iterative method, which I did not. So it does reach max recursion if I give a really long nested clause.

k) It was difficult to reconstruct the premises and conclusions because of the aforementioned error in f.

l) I also switched to “Sets” for maintaining a working set to discard / find a union of new-clauses that would automatically increase the search space.

m) Pure literals can cause a problem which I actually tested because - even though clauses with pure literals may not be resolvable. The **unprovable.cnf** contains a pure literal **D(x)**.

n) Lot of my code is in the google colab notebooks - jupyter lab does not work remotely on the agate esp. for debugging this assignment. I created multiple versions of each one of the optimizations -

   i) iterative unify vs recursive unification,

ii) History dictionary vs history list vs set
iii) KB sets vs clause lists vs KB dictionaries
iv) Memoization
v) Factoring was helpful (but I am not sure if we are expected to use it) but it is tricky so I did not include it. But since I am selecting shorter to longer clauses (by depth), it seemed better than factoring.

Resolution and resolve were modified at 9:59pm and have not been tested (as they have some bug fixes). Presently run.sh does not return any results.

2. What can you say about the time and space complexity of your program?
   W.r.t to time, only one pair at a time is resolved, when there are multiple complementary literals and two statements can get illegal resolution depending on the sequence of resolution. The order matters. When we do hand-solve, we have some clues to make it more general, but it is difficult to stop the prover as the nested functions for a single predicate can grow. The worst case size can grow exponentially and the prover can run forever.
   Size of history increases as the resolution explores all possible combinations. Space depends on the number of clauses and complexity of the clauses (depth of nested clause). We did not consider equality and it seems that it reduces the number of possibilities or paramodulation possibilities for this specific use case of considering equality.

3. What suggestions do you have for improving this assignment in the future?
   Honestly, I need to rest a bit. I thought this homework assignment would be great if we had a skeleton of clausal forms as a class and if we could instantiate it someway. Such a class clausal form, class clause, class predicates, functions for string to clause and clause to string conversion. And if there was a Node(graph) data structure - we could experiment with DFS and other search algorithms. Implementing iterative unification for a z3 prover would be easier as we have built in classes that can adapt to predicates, clauses, terms, term lists.

   In 2022, when I implemented this, it took me a course project but I implemented it in like 10-days time (there is a lot to test and none of it is trivial). This assignment would be better suited for 10 days. This could be the best improvement for this assignment.