Anirudh Narsipur

Project 1

CSCI 2951O – Spring 2022
February 25, 2022

1 Overview

This project for me was mainly an exercise in learning how to write efficient code. Although I wished to implement CDCL due to time constraints imposed by other courses did not allow me to do so.

Throughout the duration of this project I probably spent about 40-50 hours on this project. Below I briefly discuss the key aspects of my code and my personal observations. I have included a brief README laying out the code.

2 Choice Of Language

Using Julia was very nice. It was nice to use pythonic syntax and get very fast code. Julia also checks type assertions so that was nice.

3 Memory

The most critical thing for perfomance was reducing memory allocation (and deallocation). For example one of the instances initially throughout it's runtime allocated and deallocated about 240 GB of memory. A major culprit was something seemingly innocous:

$$\text{any(map}(x \to x == \text{st,ls}))$$

The above line was meant to check whether any literal in an array was in a certain state (Satisfied, Conflict, Undecided). However here the map functions allocates a new array to store the result every time. Changing this to a simple for loop drastically reduced memory consumption. There were a ton of similar optimizations like this.Generally writing C like code was necessary. For example any function that repeatedly creates array curries the array and returns the function to prevent repeated allocation. Functions that cannot do that have an array passed to them to use.

3.1 Watched Literals and Backtracking

Even though I did not implement CDCL I implemented watched literals. This was very useful as my clause set was immutable. In order to avoid the cost of mutating the clause set the clause set is immutable.Instead the only thing that changes is the assignment dictionary that maps each variable to it's assignment (Positive,Negative, Undecided). Thus watched literals allowed for quickly skipping already satisfied clauses.

All variable assignments are also stored to a stack (which is a dynamic array (size doubles each time)). Upon backtracking items from the stack are popped and assignments are undone.

4 Var to Clause Mapping

In order to avoid going through the entire clause set for unit propagation I maintain a mapping from variables to the clauses they appear in (Seperate for Positive and negative literal). Thus at each call to dpll only the clauses for the recently guessed variable are checked and then unit propagation proceeds similarly.

5 Variable Picking Heuristic and Recalcuation

The variable picking heuristic was the most significant factor. I used the Jerslow Wang heuristic which generally worked well. The variable calculating order is calculated once in the beginning and the recalculated if there are 2 guesses without any unit propagation. (2 worked best from experimentation). Updating variable picking every n recurences by itself did not make any difference but updating in the lack of unit prpgations made a tremendous difference.

5.1 Early Stopping

If there are 2 guesses without any unit propagation I also check if all clauses are satisfied without all variables being assigned in which case the algorithm stops early.

6 Pure Literal Elimination

Pure Literal Elimination did not help. Hence for BCP only unit propagation is performed. I use pure literal elimination only once before recurrence.