Write-up

Daniel McCormick and Aditya Arora

Tseitin Transform

The biggest challenge in this problem was handling the parsing of the clause. The troubles in this problem arose from mismatched brackets and other operator errors.

- Mismatched brackets were detected using a stack when pre-processing the file for errors
- Double operators were also detected in the pre-processing stage

A 3rd party, symbolic library was used to make the process of dealing with parse trees easier in python, and not have to do it from scratch. Once the clause was parsed and Tseitin had been run over it, the only real challenge was reducing the parse tree to its CNF form.

Our output was verified by using the libraries "equivalent" function to ensure that no error was made in our transformation process.

DPLL

// TODO

Our output was verified by running it on CNF files found online, as well as Prof. Ward's CNF files from ECE108.

Klee (Bugs in `coreutils`)

The first few tutorials were really interesting, and taught me some really valuable lessons about the ways `klee` could be used.

- In tutorial 3, it was *really* fascinating to see `klee` generate the test cases that would break through pseudo-walls in the "maze"
- Moreover, Tutorial 4 where a password is cracked by running `klee` on the "checker" really made me aware of that fact that just complexifying code for humans, does not make it any difficult for computers to see right through the facade

The biggest challenge in this problem was actually the compilation of `klee`. The first few parts of the tutorial worked fine with the DOCKER container version of `klee`, **but** the actual `coreutils` parts of the assignment required compilation of `klee` from scratch on a virtual machine.

The number of hours spent on actually getting `klee` to compile are not measurable on one hand. A parallely running web-server had to be taken down because the compilation was capping out the RAM on the relatively cheap VM. Optimization had to be done for storage on the VM because all the installation repositories along with build files, capped out the storage as well.

Once the compilation was done, the actual "testing" of `coreutils` was actually extremely easy.

- `md5sum`: 1 ptr error
- `mkdir`: 1 model error, 1 ptr error
- `paste`: 10 ptr errors

Encode Graph Vertex Cover to CNF

This was the only problem that required some of our own thinking. After some discussion, the ripple adder solution was tried, but sadly we lacked enough context to be able to use the bits of our output and feed it into our comparator. An earlier version with this failed attempt is linked here

In the end, we resorted to using an exponential ($n!$) growth algorithm by generating all the possible combinations of sum and simplifying to get our sum. The simplified result did reveal some insights on how we could optimize our algorithm to reduce time complexity, but the approach was not implemented due to lack of time.

Our final results for this algorithm were compared with the output of a more traditional vertex cover algorithm.