Lab 05: SAT Solver, Part 2
COSC 290 - Fall ’21

1 Overview

In this lab, you will continue evaluating satisfiability for CNF propositions. The goal is to improve last week’s algorithm using the "stop early" optimization, making your implementation runs faster with fewer recursive calls.

In addition to producing a correct result, you must implement the optimization below to receive full credit. A provided benchmark will showcase the performance gains once the optimization is properly implemented.

2 The Benchmark

Provided to you in this week’s Lab05Tester is a benchmark which runs your satisfiability algorithm against a series of propositions of increasingly complex ratios (see below). Also provided is PropositionGenerator, which is used by the benchmark to randomly generate CNF propositions.

This benchmark reports back the algorithm’s resulting:

- cost, meaning the number of recursive calls
- average time, the average wall-clock time (in milliseconds) it took to determine the satisfiability of the series of propositions of the respective ratio

Below is an example output of the benchmark – the before uses the algorithm from last week’s lab, and the after implements the "stop early" optimization on the following page. Bear in mind – your results won’t match this example exactly (everyone’s computer and implementation will be a little different), but you will see a significant improvement:

Before:

<table>
<thead>
<tr>
<th>Ratio r</th>
<th>fraction sat</th>
<th>avg. cost</th>
<th>avg. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1352.9</td>
<td>13.6</td>
</tr>
<tr>
<td>2.0</td>
<td>1.0</td>
<td>30709.8</td>
<td>264.5</td>
</tr>
<tr>
<td>3.0</td>
<td>1.0</td>
<td>195743.8</td>
<td>2088.9</td>
</tr>
<tr>
<td>4.0</td>
<td>1.0</td>
<td>658794.6</td>
<td>9681.2</td>
</tr>
</tbody>
</table>

Implementation timed out!

After:

<table>
<thead>
<tr>
<th>Ratio r</th>
<th>fraction sat</th>
<th>avg. cost</th>
<th>avg. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>48.9</td>
<td>3.1</td>
</tr>
<tr>
<td>2.0</td>
<td>1.0</td>
<td>408.0</td>
<td>8.7</td>
</tr>
<tr>
<td>3.0</td>
<td>1.0</td>
<td>972.5</td>
<td>21.4</td>
</tr>
<tr>
<td>4.0</td>
<td>1.0</td>
<td>2186.8</td>
<td>68.1</td>
</tr>
<tr>
<td>5.0</td>
<td>0.2</td>
<td>3835.4</td>
<td>126.5</td>
</tr>
<tr>
<td>6.0</td>
<td>0.0</td>
<td>2789.8</td>
<td>102.7</td>
</tr>
<tr>
<td>7.0</td>
<td>0.0</td>
<td>2236.4</td>
<td>97.6</td>
</tr>
<tr>
<td>8.0</td>
<td>0.0</td>
<td>1507.8</td>
<td>77.7</td>
</tr>
</tbody>
</table>

The benchmark is based on a famous scientific discovery – In 1994, researchers learned two important properties about CNF propositions (which were reported in Science). First, the likelihood that a proposition is satisfiable depends
on the ratio between its number of clauses and variables.

When the ratio is less than 4, almost all propositions are satisfiable; when the ratio is above 5, almost all are unsatisfiable. Second, the hardest propositions to check have ratios between 4 and 5.

3 Your Task

Your task is to improve your satisfiability algorithm from by implementing the "stop early" optimization below:

3.1 "Stop Early" Optimization Overview

As you are building a truth assignment via recursion, it may not be necessary to assign every variable a value to determine whether the truth assignment leads to satisfiable result.

For example, suppose \( \varphi := (p \lor q) \land (p \lor r) \land (p \lor s) \) and truth assignment \( T \) has assigned \( p \) to True. In this case, we can already tell \( \varphi \) is satisfiable regardless of how \( q, r, \) or \( s \) are assigned.

Another example, suppose \( \varphi := (p) \land (q \lor r) \land (s \lor \neg q) \) and the truth assignment \( T \) has assigned \( p \) to False. In this case, \( \varphi \) is unsatisfiable under \( T \) regardless of how \( q, r, \) and \( s \) are assigned.

One final example, suppose \( \varphi := (\neg p \lor q) \land (\neg p \lor r) \) and truth assignment \( T \) has assigned \( p \) to True. In this case, we do not know yet whether \( \varphi \) is satisfiable or not – it may or may not be depending on how \( q \) and \( r \) are assigned.

3.2 Implementing the Optimization

You will implement the "stop early" optimization by checking the following on each step of the recursive algorithm, regardless of if every variable in the proposition has an assigned value in the truth assignment:

- if you have sufficient variable assignments to determine every clause to be True, stop early and return True.
- if you have sufficient variable assignments to determine at least one clause False, stop early and return False.
- if neither of the above scenarios are realized, the proposition is inconclusive given this (partially incomplete) truth assignment. Therefore, you follow the same recursive case as in last week’s Part 1:
  1. select an unassigned variable from the truth assignment and assign it true.
  2. recursively check if the proposition is satisfiable with this new truth assignment. If not, assign the same variable false and recursively check the satisfiability again
  3. if the proposition is not satisfiable with the selected variable assigned true or false, return false.

In CNFProposition.java, implement the isSatisfiableOptimizedHelper method with an enhanced version of the satisfiability algorithm you wrote last week, incorporating the stop early optimization.

4 Pre-Lab Questions

After reading this document and provided code, answer the following before we meet (we will discuss in-lab):

1. Which of the below Java conditionals could potentially throw a NullPointerException?:

   (a) if (x.equals(y) || x == null)
   (b) if (x == null || x.equals(y))
   (c) if (x == null && x.equals(y))
   (d) if (x != null && x.equals(y))

2. Suppose we had a static helper function, evaluateClause, which attempts to evaluate a single clause given a (potentially incomplete) truth assignment. What would be an ideal return type for this function?
5 Submission

See the top of this document for your lab section’s due date/time. When uploading your submission to Moodle submit only the files listed below:

- CNFProposition.java
- Lab05Tester.java