CDCL Based SAT-Solver

Kevin Wu
1600012832

July 16, 2017

1 Clarification

Despite the different algorithm, there are still many similarities between DPLL and CDCL. Many of the fundamental implementation detail are stated in **DPLL.pdf**. Here, I just specify several difference.

I followed `https://en.wikipedia.org/wiki/Conflict-Driven_Clause_Learning` instructions to implement this SAT-Solver.

2 Key Implementation Ideas

1. **Main Process:** Quoted from Wikipedia

   Conflict-Driven Clause Learning

   (a) Select a variable and assign True or False. Remember the assignment.
   (b) Apply Boolean constraint propagation (Unit propagation).
   (c) Build the implication graph.
   (d) If there is any conflict then analyze the conflict and non-chronologically backtrack ("back jump") to the appropriate decision level.
   (e) Otherwise continue from step 1 until all variable values are assigned.

2. **Set Pure:** Based on the instructions from Wikipedia, there is no *Set Pure* procedure. But this process does no harm to the other parts; so I added this feature in my code like DPLL version.
3. **Implication Graph:** This graph (DAG) shows how the program determines every literals step by step. Basically, when you do *BCP* and *Set Pure*, add a deduce edge; when you do *Decision*, add a decision edge. This graph is essential in *Conflict Analysis*.
4. **Conflict Analysis:** This is how program learns to analyze the reason why conflict occurs. Basically, you go from conflict point and back search all the decision points that contribute to this deduction, thus the clause learned. And then add it to the original clause group.

   Since it is a DAG, in my implication way, I implicitly build the graph, and store this clause when literal was deduced.
5. **Back Jump:** From the learned clause, the program can jump back to the second latest decision point, for the last one can be determined by BCP after the second latest decision.

3 Potential Traps

1. It is worth mentioning that due to the learned clause, you will have to consider how to deal with those eliminated literals (in *Decision*) in the new clause. Because my implementation needs to guarantee every time equation list can not have eliminated literals, I have to maintain the stack manually and insert those literals as eliminated in previous stack level and recover them in recursion.
2. Since *Set Pure* preserves the satisfiability, those literals eliminated by *Set Pure* do not need to draw into *Implication Graph*.

Note: Aside from CDCL based SAT-Solver, I also implemented a DPLL based SAT-Solver, the code and report of which can be found in `https://github.com/Shlw/Naive-SAT-Solver`