

CS357 Homework 2

This is due Tuesday, Nov 8. Please give me (Dill) the solutions to the written problems, and email me the results of your BDD SAT implementation.

1. Prove the following algebraic properties: The proofs can be done either by the “geometric interpretation” of constrain, or by induction on number of variables in the BDD. Choosing the right approach to your proof can make the it simple.

(a) $(g \wedge (f \downarrow g)) = (f \wedge g)$

(b) $(f \wedge g) \downarrow h = (f \downarrow h) \wedge (g \downarrow h)$. (In fact, \downarrow distributes over all Boolean operators.)

2. Shannon decomposition works with generalized cofactor as well as cofactoring on a single variable. In other words, $f = \text{ite}(g, f \downarrow g, f \downarrow \neg g)$. Prove it (Hint: we used the lemma $(g \wedge (f \downarrow g)) = (f \wedge g)$, above).
3. Find two functions f and g where the BDD for $f \downarrow g$ is bigger than the BDD for f .
4. Build a CNF solver using constrain as discussed in lecture. I.e., given a vector of BDDs, (f_1, f_2, \dots, f_n) , repeatedly constrain the tail of the list by the first BDD in the list until one of the BDDs becomes 0 or until there is only one (nonzero) BDD remaining.

Try it on some CNF examples (creating a vector of BDDs, one per clause) and see how well it does. Report which examples you did, how long it took and the result (time out, out of memory, sat, unsat). The goal of this is to give you some basic experience with BDDs. The minimum effort required is to get it to run on some examples.

There is no upper bound to the effort you can expend on this if you want to. You may even find a better SAT algorithm! The first obvious improvements are to do early quantification, merge groups of clauses, and trying to order the clauses strategically (all of these would probably be based on finding clauses with many common variables).

Compare with building a single BDD by ANDing the f_i 's together. Please run it on some of the SAT problems you generated in the previous homework. Also, pick a hard example out of this set:

<http://www.cril.univ-artois.fr/SAT11/results/globalbybench.php?idev=45&idcat=62>

A 10 CPU-minute timeout is ok (but you may set it longer if you are curious). After every k iterations of constraining everything in the vector by the first remaining BDD in the vector, print k the number of nodes in the biggest BDD in the vector. Choose k be reasonable as a function of the number of clauses.

Please email me (dill@cs.stanford.edu) a report of results on at least three SAT problems of various degrees of difficulty. Describe what you did,

especially any non-obvious implementation. For each SAT problem, report the outcome (satisfiable, not satisfiable, time-out, memory-out or crash, largest BDD reported, maximum number of iterations of constrain).

There are lots of BDD packages available. My experience is with CUDD, which is widely used. However, I had trouble building the available version, probably because of a missing subdirectory. I *did* get a working version very recently by downloading the PerlDD package (which has CUDD embedded in Perl) and adding `#define UNIX 1` right after the includes in `util/pipefork.c`.

For fun, you may want to try the DDcal-0.8 packages which interactively draws BDDs using “dot”. It uses PerlDD.