Alexander J. Summers

Program Verifiers and Program Verification

Exercise Sheet 1: SAT Solving Algorithms

Assignment 1 (Tseitin CNF Conversion)

The conversion of a formula to CNF shown in the lecture slide "Conversion to CNF" can increase the size of the formula exponentially.

1. Which of the steps shown on the lecture slide increase the size of the formula?
2. Write down an example input formula whose CNF form according to these rules is indeed exponentially larger.

An alternative approach for converting to CNF (Tseitin, 1968) is to introduce *additional variables* to replace subformulas which would otherwise be duplicated. In particular, we can avoid the duplication caused by distributing disjunctions over conjunctions, by introducing *extra variables* in the rewritten formula. We achieve this as follows:

First, apply the first four rules from the slide, so that we are dealing only with a combination of conjunctions and disjunctions over literals. Now, we process the formula bottom-up as follows: choose a subformula which is either a conjunction or a disjunction of *two literals* (i.e. a smallest subformula involving a boolean connective). Let's suppose it is a conjunction, say $p \land q$. Pick a fresh propositional variable to represent this subformula, say x, and replace the subformula with x, recording that x represents the subformula $p \land q$. Note that this has made the original formula smaller; repeat this procedure, recording the mappings between fresh variables and subformulas.

Now, take the resulting formula, and for each fresh variable introduced during the above process (e.g. x for the subformula $p \land q$) conjoin clauses representing the constraint that $x \Leftrightarrow (p \land q)$ must be true. For such a conjunction subformula, the appropriate clauses are $(\neg x \lor p) \land (\neg x \lor q) \land (x \lor \neg p \lor \neg q)$. Note that exactly three clauses are needed per subformula processed in this way; the size of the resulting formula will be linear in the size of the original formula.

3. What are the appropriate clauses to add if the fresh variable replaces a *disjunction* of two literals (say y replaces $p \lor q$)?
4. When should processing the original formula (and introducing new variables) be terminated?
5. For the example you wrote in part 2, what is the result of this new CNF transformation?

6. The runtime of a general SAT solver is (in the worst case) accepted to be exponential in the number of variables involved. Why does the introduction of additional variables here not cause a corresponding performance problem?

Assignment 2 (Applying SAT algorithms)

Consider the example formula (similar to the one shown on the "DPLL Implication Graph" slide):
$(n \lor p)\land$
$(\neg n \lor p \lor q)\land$
$(\neg n \lor p \lor \neg q)\land$
$(\neg p \lor r)\land$
$(\neg u \lor t)\land$
$(\neg r \lor \neg s \lor t)\land$
$(q \lor s)\land$
$(\neg p \lor t \lor u)\land$
$(\neg p \lor \neg t \lor \neg u)\land$
$(\neg r \lor \neg t \lor u)$

1. Apply the DPLL algorithm to this example. When splitting on decision literals, follow this initial order: n true, p true, s false, t true.
2. Apply the CDCL algorithm to this example, using the same initial order of decision literals. What differences do you observe?

Assignment 3 (Correctness of DPLL)

Consider the DPLL algorithm presented in the lectures (not the CDCL extension).

1. Prove that the algorithm terminates on any input.
2. Prove that, if the algorithm returns $(\texttt{sat}, M)$, then the model M is a model for the original input formula.