Theorem Proving, January–April 2014

Assignment 2, 15 April 2014
Due Friday, 2 May 2014

Instructions for submitting solutions

Please submit your solutions electronically by email to madhavan@cmi.ac.in *to facilitate evaluation.*

Preferrably, send an electronic document in PDF (you can generate it in LaTeX or OpenOffice or Word or whatever, but send only PDF). If you can't do this, send scanned copies of handwritten pages.

The numbers quoted with each question refer to the book *Decision Procedures: An Algorithmic Point of View* by Daniel Kroening and Ofer Strichman. Check if you think there are typos!

1. **Exercise 2.9 (polynomial-time (restricted) SAT)**

   Consider the following two restrictions of CNF:

   - A CNF formula in which there is not more than one positive literal in each clause.
   - A CNF formula in which no clause has more than two literals.

   (a) Show a polynomial-time algorithm that solves each of the problems above.

   (b) Show that every CNF can be converted to another CNF which is a conjunction of the two types of formula above. In other words, in the resulting formula all the clauses are either unary, binary, or have not more than one positive literal. How many additional variables are necessary for the conversion?

2. **Problem 2.12 (incremental satisfiability)**.

   Given two CNF formulas C_1 and C_2, under what conditions can a conflict clause learned while solving C_1 be reused when solving C_2? In other words, if c is a conflict clause learned while solving C_1, under what conditions is C_2 satisfiable if and only if $C_2 \wedge c$ is satisfiable? Try to get as general, i.e., weak, set of conditions on C_2 relative to C_1 and c. How can the condition that you suggest be implemented inside a SAT solver?

   Hint: Think of CNF formulas as sets of clauses.

3. **Problem 2.14 (implementing Apply with *ite*)**

   Efficient implementations of BDD packages do not use Apply; rather they use a recursive procedure based on the *ite* (if-then-else) operator. All binary Boolean operators can be expressed as such expressions. For example,

   $$f \vee g = ite(f, 1, g), \quad f \wedge g = ite(f, g, 0),$$
   $$f \oplus g = ite(f, g, g), \quad \neg f = ite(f, 0, 1).$$

   How can a BDD for the *ite* operator be constructed? Assume that x labels the root nodes of two BDDs f and g, and that we need to compute $ite(c, f, g)$. Observe the following equivalence:

$$ite(c, f, g) = ite(x, ite(c|_{x=1}, f|_{x=1}, g|_{x=1}), ite(c|_{x=0}, f|_{x=0}, g|_{x=0})).$$

Hence, we can construct the BDD for $ite(c, f, g)$ on the basis of a recursive construction. The root node of the result is x, $low(x) = ite(c|_{x=0}, f|_{x=0}, g|_{x=0})$, and $high(x) = ite(c|_{x=1}, f|_{x=1}, g|_{x=1})$. The terminal cases are

$$ite(1, f, g) = ite(0, g, f) = ite(f, 1, 0) = ite(g, f, f) = f,$$
$$ite(f, 0, 1) = \neg f.$$

(a) Let $f := (x \wedge y)$, $g := \neg x$. Show an *ite*-based construction of $f \vee g$.

(b) Present pseudocode for constructing a BDD for the ite operator. Describe the data structure that you assume. Explain how your algorithm can be used to replace APPLY.
