Logic Synthesis & Verification, Fall 2018

National Taiwan University

Problem Set 2

Due on 2018/10/31
Drop your solution in the instructor's mailbox in EE2 Building by 18:00

1 [Cofactor]

(10%) Given two Boolean functions f and g and a Boolean variable v, prove the following equalities.

(a) (5%) $(\neg f)_v = \neg(f_v)$, and
(b) (5%) $(f \oplus g)_v = (f_v) \oplus (g_v)$.

2 [Quantification]

(20%)

(a) (4%) Consider the following 8 quantified Boolean formulas

$$\begin{aligned}
F_1 &: \ \exists x, \exists y. f(x, y, z), \\
F_2 &: \ \exists y, \exists x. f(x, y, z), \\
F_3 &: \ \exists x, \forall y. f(x, y, z), \\
F_4 &: \ \forall y, \exists x. f(x, y, z), \\
F_5 &: \ \forall x, \exists y. f(x, y, z), \\
F_6 &: \ \exists y, \forall x. f(x, y, z), \\
F_7 &: \ \forall x, \forall y. f(x, y, z), \\
F_8 &: \ \forall y, \forall x. f(x, y, z).
\end{aligned}$$

List the set of implications $F_i \rightarrow F_j$ for $i, j = 1, \ldots, 8$ and $i \neq j$.

(b) (4%) Prove or disprove

$$\exists x, \forall y.(f(x, y) \wedge g(y)) = \forall y.((\exists x.f(x, y)) \wedge g(y)).$$

(c) (4%) Prove or disprove

$$\forall y, \exists x.(f(x, y) \wedge g(y)) = \forall y.((\exists x.f(x, y)) \wedge g(y)).$$

(d) (4%) Prove or disprove

$$\forall x.(f(x, y) \wedge g(x, y)) = (\forall x.f(x, y)) \wedge (\forall x.g(x, y)).$$

(e) (4%) Prove or disprove

$$\exists x.(f(x, y) \wedge g(x, y)) = (\exists x.f(x, y)) \wedge (\exists x.g(x, y)).$$

3 [BDD and ITE]

(10%) Let $f = a \oplus b \oplus c \oplus d$ and $g = b \wedge d$.

(a) (5%) Draw the edge-complemented ROBDDs of f and g with variable ordering $a < b < c < d$ (a on top).
(b) (5%) Let the ROBDDs of f and g be F and G, respectively. Compute COMPOSE(F, b, G).

4 [BDD Onset Counting]

(10%) Design a linear-time algorithm that counts the number of onset minterms of a given ROBDD.

5 [ZDD]

(10%) Construct a ZDD that represents the set of self-avoiding paths over the edges connecting the top corner to the lower right corner in the graph of Figure 1. (E.g., acgi is a self-avoiding path, but bcegh is not.)

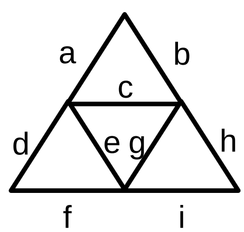

Fig. 1. Graph for self-avoiding path enumeration.

6 [SAT Solving]

(20%) Consider SAT solving the CNF formula consisting of the following ten clauses

$$C_1 = (b + d), C_2 = (a + b + c' + d'), C_3 = (a + b' + c), C_4 = (a' + b' + d),$$
$$C_5 = (a + b' + c'), C_6 = (a + c' + d), C_7 = (a' + b + c), C_8 = (a + b + c),$$
$$C_9 = (b' + c + d'), C_{10} = (a' + b' + c' + d').$$

(a) (10%) Apply implication and conflict-based learning to solve the above CNF formula. Assume the decision order follows a, b, c, and then d; assume each variable is assigned 0 first and then 1. Whenever a conflict occurs, draw the implication graph and enumerate all possible learned clauses under the Unique Implication Point (UIP) principle. (In your implication graphs, annotate each vertex with "`variable = value@decision_level`", e.g., "$b = 0@2$", and annotate each edge with the clause that implication happens.) If there are multiple UIP learned clauses for a conflict, pick the one with the UIP closest to the conflict vertex in the implication graph.

(b) (10%) The **resolution** between two clauses $C_i = (C_i^* + x)$ and $C_j = (C_j^* + x')$ (where C_i^* and C_j^* are sub-clauses of C_i and C_j, respectively) is the process of generating their **resolvent** $(C_1^* + C_j^*)$. The resolution is often denoted as

$$\frac{(C_i^* + x) \qquad (C_j^* + x')}{(C_1^* + C_j^*)}$$

A fact is that a learned clause in SAT solving can be derived by a few resolution steps. Show how that the learned clauses of (a) can be obtained by resolution with respect to their implication graphs.

7 [SAT Solving]

(20%)

(a) (10%) Write a CNF formula stating the pigeon-hole problem: *There are n holes and $n + 1$ pigeons. Every hole accommodates at most one pigeon and every pigeon must be in some hole.*

(b) (10%) Use MiniSAT (http://minisat.se/) to solve the pigeon-hole problem for $n = 4, 5, 6$. (Note that the formulas should be in the DIMACS format http://www.satcompetition.org/2009/format-benchmarks2009.html.) Print out the MiniSAT statistics. Do you expect the solver is scalable on this problem? Why or why not?