Logic Synthesis & Optimization

Spring 2019, Homework # 4
Due Date: Monday, April 1.

1) **Kernel Extraction (10 points):** Consider the Boolean Function $F = uvy + vwy + xy + uz + vz$.
   - Identify all co-kernel/kernel paris of F. State their levels.
   - Use the two different methods that we studied in class to compute the co-kernel/kernel pairs. One method is given in the textbook based on the cube-intersection matrix, and the 2nd one given in the slides based on the recursive `compute_kernel()` procedure.
2) **Weak Algebraic Division (10 points):** We have studied an algorithm to perform weak algebraic division in class (also given in the text) that can be used to decompose a function F, algebraically, as $F_{dividend} = G_{divisor} \cdot H_{quotient} + R_{remainder}$. Divide $F = ab + ac + ad' + bc + bd'$ by the following:
   - $G = a + b$. What is the quotient and the remainder?
   - $G = c + d'$. What is the quotient and the remainder? Note: You should show how the "algorithm" works; i.e. follow the pseudo-code given in Fig. 10.9.
3) **Multi-Level Synthesis (20 points):** Consider a circuit with 6 inputs (u, v, w, x, y, z) and 2 outputs (f, g), given as: $f = uvy + vwy + xy + uz + vz$ and $g = uxy + xz + uvz + vwz + yz$. You are asked to perform multi-level logic synthesis to reduce the cost of the implementation. Your operations may include any number of iterations of the following operations: i) co-Kernel and Kernel identification; ii) common sub-expression extraction; iii) resubstitution; iv) factorization and/or (espresso-type) simplification, if you like. The objective is to get a low-cost multi-level optimized form with minimum number of literals. Show the sequence of operations that you perform, and draw the final multi-level logic network.

   [Of course, multi-level logic synthesis isn't unique, and there isn't a well-defined notion of minimality in multi-level logic synthesis. So there might be multiple solutions. I want to see how you approach the problem. BTW, we've performed this set of procedures in class, so this should not be an alien concept.]
4) **20 points - Bi-Decomposition:** Let $F = ab + ad + cd$, and $G = a + c$ be two Boolean functions. Can the function F be decomposed as:
   - $F = G \cdot H$. If so, prove it and derive H. If not, then show why it is not possible.
   - $F = G + H$. If so, prove it and derive H. If not, then show why it is not possible.
   - $F = G \overline{\oplus} H$. If so, prove it and derive H. If not, then show why it is not possible.

- Note: feel free to study the paper on *BDS: BDD-Based Logic Decomposition* by Yang and Ciesielski, given on the class website. You may use K-maps or ROBDDs to solve this problem.

5) **BDD based Bi-Decomp (20 points)**. Using the variable order $a > c > b > d$, construct an ROBDD for the Boolean function F given above as $F = ab + ad + cd$. Using this variable order and the AND Bi-decomposition approach of Yang and Ciesielski, is it possible to identify $G = a + c$ to perform the decomp $F = G \cdot H$? If so, identify a corresponding H. If not, then identify a suitable G, H.
6) **Boolean versus Algebraic Division (10 points):** This exercise will show you the power of Boolean division *viz-a-viz* algebraic division. Consider the Boolean function $F = ab + acd + ac'd' + a'b'c + a'b'd$. We know that AND bi-decomposition ($F = G \cdot H$) is a special case of division where: i) Remainder is zero; and ii) G and H may have overlapping support (hence Boolean division). Let $G = a + b'$. Do the following:

   - Perform the AND bi-decomposition $F = G \cdot H$. In other words, identify H using Boolean decomposition principles.
   - Now, perform the weak algebraic division of F by G.
   - What do you observe as far as the capabilities and limitations of the Boolean and algebraic procedures?

7) **Experiments with SIS (10 points):** In this question, you are asked to perform experiments with SIS, and observe in a step-by-step fashion how synthesis using the algebraic model operates. SIS takes as input a circuit netlist (in blif format) or a set of Boolean equations (in eqn format), or a PLA (in *.pla format). For minimal area optimization, important operations used are: i) node collapse/eliminate; ii) kernel extraction; iii) node re-substitution; and iv) node simplification using espresso. Follow the steps below to understand these operations that we have also studied in class. Observe the effect of the operations and describe them in your own words. Print out and attach your 'synthesis run' to your submission.

   - From the class web-page download the file: hw5.eqn. The file represents a Boolean network with three outputs F, G, H. Read the file in SIS: 'read_eqn hw5.eqn'.
   - What is the cost of implementation (in number of SOP literals) of this network? Verify your answer by running the command: print_stats.
   - Run the command 'resub -a'. This command performs algebraic re-substitution. Observe the effect of re-substitution by writing out the modified network: just type write_eqn. What do you think has happened?
   - Now type 'fx'. This performs kernel extraction. Write out the network again (write_eqn) and

identify the kernel(s) that was extracted. Also, do you observe any redundancy in the network? Can kernel extraction create an redundancies?

- Can you eliminate the redundancy by simplifying the network? Run command: ’simplify’ and then ’write_eqn’ and see for yourself.
- Now observe the importance of the “sweep” command: run sweep and then write_eqn. What do you think sweep tries to do?
- Now print_stats to see if the cost of the implementation has reduced. What is the number of SOP literals now?
- Of course, you can destroy your factorization, substitution and simplification by collapsing the network. Try: collapse; print_stats; write_eqn.
- From this collapsed network, perhaps you can extract more common sub-expressions (cubes or kernels)? Try: ’fx’ again and see what happens. Now do ’simplify’, ’sweep’, ’resub -a’ again and observe and write in your own words what’s happening.
- That’s multi-level logic optimization for you!