CSE 311: Foundations of Computing I

Homework 1 (due Monday, October 10th at 11:00 PM)

Directions: *Write up carefully argued solutions to the following problems. Each solution should be clear enough that it can explain (to someone who does not already understand the answer) why it works. However, you may use results from lecture, the reference sheets, and previous homeworks without proof.*

1. Watch Your Language (18 points)

Translate these English statements into logic, making the atomic propositions as simple as possible and exposing as much of the logic via symbols as possible.

(a) [4 Points] Define a set of *at most four* atomic propositions. Then, use them to translate "If we can cover 20% of the Sahara desert with solar panels and the rest with wind turbines then the desert will not expand and we can generate four times as much electricity as the entire planet consumes right now."

(b) [8 Points] Define a set of *at most four* atomic propositions. Then, use them to translate each of these sentences:

i) If the stack is empty, you can push but not pop.

ii) If the stack is full, you can pop but not push.

iii) If the stack is neither full nor empty, you can both push and pop.

(c) [6 Points] Define a set of *at most four* atomic propositions. Then, use them to translate each of these sentences:

i) Your program uses either a `LinkedList` or an `ArrayList` to store a sequence of objects.

ii) Your program will perform well if it uses a `LinkedList` and an iterator to access the list elements, but if the program doesn't use an iterator to access the list elements it will perform poorly unless it uses an `ArrayList`.

2. Doublespeak (12 points)

Consider the following sentence: If we have lecture today and James is out of town, Kevin will teach lecture.

(a) [4 Points] Define a set of *at most three* atomic propositions. Then, use them to translate the sentence above into propositional logic.

(b) [3 Points] Take the contrapositive of the logical statement from part (a). Then, simplify it so that all $\neg$ symbols are next to atomic propositions.

(c) [3 Points] Translate the sentence from part (b) back to English.

(d) [2 Points] Must your English sentence from part (c) have the same truth value the original English sentence above? Why or why not?

3. Circuit Breaker (14 points)

(a) [2 Points] Write a table showing the values of the boolean function $B(r, s, t)$ defined as follows. If $r = s = 1$, then $B(r, s, t) = \neg t$. If $r = s = 0$, then $B(r, s, t) = 1$. Finally, if exactly one of r and s is 1, then $B(r, s, t) = s \wedge t$.

(b) [8 Points] Draw the diagram and a table of values for two different circuits that use exactly one OR gate and one XOR gate (and nothing else).

The table of values should show the values on each wire in the circuit (not just the inputs and outputs). The two circuits must be different enough that the number of 1s in the output columns are not the same.

(c) [4 Points] By adding a single NOT gate to a circuit with exactly one OR and one XOR (possibly one of the circuits you drew in part (b)), we can make a circuit that calculates the function B from part (a). Give the circuit **and its table of values** (including all wires, not just input and output), the last column of which should match your answer from part (a).

Hint: There are only four places where a NOT gate can be added to a circuit with just one OR and XOR, so there are only a small-ish number of possible circuits to test out. One of them will work.

4. Short Circuit (14 points)

(a) [2 Points] Write a table showing the values of the boolean function $A(r, s, t)$ defined as follows. If $r = 1$, then $A(r, s, t) = s \to t$, and if $r = 0$, then $A(r, s, t) = t$.

As always, you should include intermediate expressions (in this case, $s \to t$) as columns in your table.

(b) [4 Points] If we consider circuits with one input, a, that use only a single A gate, there are still $3^3 = 27$ different possibilities because each of the three inputs of the A gate can be either T, F, or a.

Find one of those circuits that calculates the same value as $\neg a$. Describe the circuit either by drawing it or writing it as an expression. Then, write a table showing that its values match those of $\neg a$.

Hint: To shorten your search, there is a solution where the three inputs to the A gate are exactly one of each of T, F, and a, in some order.

Note: Since there is only one variable, a, your table should have only 2 rows: one for $a = 1$ and one for $a = 0$. In each row, your circuit is calling A with known values for each input, so the reader can figure out the value just by looking at the appropriate row in your table from part (a). For that reason, your table only needs three columns: a, the expression for your circuit, and $\neg a$ for comparison.

(c) [4 Points] If we consider circuits with *two inputs*, a and b, that use only a single A gate, there are now $4^3 = 64$ different possibilities because each of the three inputs of the A gate can be either T, F, a, or b.

Find one of those circuits that calculates the same value as $a \vee b$. Describe the circuit either by drawing it or writing it as an expression. Then, write a table showing that its values match those of $a \vee b$.

Hint: To shorten your search, there is a solution where the three inputs to the A gate are a, b, and either T or F, in some order.

Note: Your table should now have four rows and four columns since we need to consider all the possible values of both a and b.

(d) [4 Points] Find a circuit, using **at most four** A gates but no other gates, that calculates the same value as $a \wedge b$. Describe the circuit either by drawing it or writing it as an expression. Then, write a table showing that its values match those of $a \wedge b$. (Include intermediate expressions as columns in your table.)

Hint: For this one, you'll want to spend more time thinking and less time trying out different combinations. Make use of what you learned in parts (b) and (c).

5. Same Difference (16 points)

For each of the following pairs of propositions, use truth tables to determine whether they are equivalent or not.

Include the full truth table and state whether they are equivalent. (In principle, only one row is needed to show non-equivalence, but please turn in the entire table so that we can give partial credit in case of errors.) Your truth table should include columns for all subexpressions.

(a) [3 Points] $(P \land Q) \land (P \lor Q)$ vs. $P \lor Q$

(b) [3 Points] $P \oplus Q$ vs. $\neg P \oplus \neg Q$

(c) [5 Points] $P \to (Q \to R)$ vs. $(P \land Q) \to R$

(d) [5 Points] $(P \to Q) \to R$ vs. $(Q \to P) \lor R$

6. Same to You (16 points)

Prove the following assertions using a sequence of logical equivalences.

Hint: For equivalences where one side is much longer than the other, a good heuristic is to start with the longer side and try to apply the rules that will shorten it. These are Identity, Domination, Absorption, Negation, and Double Negation. (My solutions, put together, use every one of those rules at least once.)

Hint for (d): if you find yourself wanting to transform ¬T into F, you may want to back up and try another approach to solving the problem. It is possible to prove ¬T ≡ F using equivalences, but it takes a bit of work to do so in cozy.

(a) [4 Points] $P \to (Q \to R) \equiv (P \land Q) \to R$

(b) [4 Points] $(\neg P \to Q) \land (Q \to P) \equiv P$

(c) [4 Points] $(P \to Q) \lor (P \to \neg Q) \equiv$ T

(d) [4 Points] $((P \lor \neg P) \to P) \land (Q \to P) \equiv P$

Submit and check your answers to this question here:

`http://cozy.cs.washington.edu`

You can make as many attempts as needed to find a correct answer.

Just to be safe, we will include the problem for 0 points on Gradescope, and you can submit your answer there if you have trouble with cozy, but cozy is where we'd like you to submit your answers

Documentation is available on the website, at the the link labelled "Docs" at the top of the page.

7. Case Closed (10 points)

Searching for a secure encryption function, you find a GitHub repo with three implementations: `A`, `B`, and `C`. The documentation in the repo tells you that **only one of the implementations is secure** — the other two are not! All three implementations are obfuscated, so you cannot tell which one is secure by examining the source. However, each implementation has a documentation sentence:

`A` "This implementation (`A`) is not secure."

`B` "This implementation (`B`) is not secure."

`C` "If implementation `B` is not secure, then implementation `A` is not secure."

Unfortunately, the documentation in the repo also tells you that **only one documentation sentence is true** — the other two are false!

Which implementation is the *secure* one? (Not necessarily the one whose documentation sentence is true!) Justify your answer by considering each possibility for which implementation is the secure one, and showing which of the sentences would be true in each case. Only one possibility should match the description above.

8. Extra Credit: XNORing (0 points)

Imagine a computer with a fixed amount of memory. We give names, $R_1, R_2, R_3, \dots$, to each of the locations where we can store data and call these "registers." The machine can execute instructions, each of which reads the values from some register(s), applies some operation to those values to calculate a new value, and then stores the result in some register. For example, the instruction $R_4 := \texttt{AND}(R_1, R_2)$ would read the values stored in R_1 and R_2, compute the logical and of those values, and store the result in register R_4.

We can perform more complex computations by using a sequence of instructions. For example, if we start with register R_1 containing the value of the proposition A and R_2 containing the value of the proposition B, then the following instructions:

1. $R_3 := \texttt{NOT}(R_1)$
2. $R_4 := \texttt{AND}(R_1, R_2)$
3. $R_4 := \texttt{OR}(R_3, R_4)$

would leave R_4 containing the value of the expression $\neg A \vee (A \wedge B)$. Note that this last instruction reads from R_4 and also stores the result into R_4. This is allowed.

Now, assuming A is stored in register R_1 and B is stored in register R_2, give a sequence of instructions that

- only uses the `XNOR` operation (no `AND`, `OR`, etc.),
- only uses registers R_1 and R_2 (no extra space), and
- ends with B stored in R_1 and A stored in R_2 (i.e., with the original values in R_1 and R_2 swapped).