CSE 311: Foundations of Computing I Spring 2024

Problem Set 1

Due: Wednesday, April 3rd by **11:00pm**

Instructions

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 homework without proof.

Collaboration policy. You are required to submit your own solutions. You are allowed to discuss the homework with other students. However, the **write up** must clearly be your own, and moreover, you must be able to explain your solution at any time. We reserve ourselves the right to ask you to explain your work at any time in the course of this class.

Late policy. You have a total of **three** late days during the quarter, but you can only use one late day on any one problem set. Please plan ahead. The final problem set will not be accepted late.

Solutions submission. Submit your solution via Gradescope. In particular:

- Submit *one* PDF file containing the solution to all tasks in the homework. Each numbered task should be solved on its own page (or pages). Follow the Gradescope prompt to link tasks to pages.
- Do not write your name on the individual pages – Gradescope will handle that.
- You are not required to typeset your solution, but your submission must be **legible**. It is your responsibility to make sure solutions are readable – we will *not* grade unreadable write-ups.

Legal Disclaimer. This homework assignment may include paid product placement or promotion.

Task 1 – Translate, Translate, For a Very Important Date [18 pts]

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) Define a set of *at most four* atomic propositions. Then, use them to translate "If the area is low and wet, then Sitka Spruce are the most common spruce in the area. Otherwise, in high or dry territory, Engelmann Spruce are the most common spruce in the area."

b) Define a set of *at most four* atomic propositions. Then, use those propositions to translate each of these sentences:

i) If the queue is empty, you can insert but not remove.

ii) If the queue is full, you can remove but not insert.

iii) If the queue is neither full nor empty, you can both insert and remove.

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

i) Either your cat has fleas or your dog has fleas.

ii) If your dog has fleas and you administer flea medicine, then your dog is happy. But if your cat has fleas and you administer flea medicine, then your dog is not happy.

Task 2 – You're In Good Hands With Translate [12 pts]

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

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

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

For the latter, you can rewrite an expression of the form "$\neg(P \wedge Q)$" as "$\neg P \vee \neg Q$" because the two expressions always have the same truth value. (They are "equivalent".) Similarly, you can rewrite an expression of the form "$\neg(P \vee Q)$" as "$\neg P \wedge \neg Q$". These fact that these pairs of expressions are equivalent are known as De Morgan's Laws. We will see more examples in Topic 2.

c) Translate the sentence from part (b) back to English.

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

Task 3 – Let's Cut to the Case [10 pts]

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`'s documentation "This implementation (`A`) is not secure."

`B`'s documentation "This implementation (`B`) is not secure."

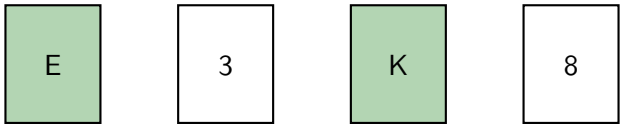
`C`'s documentation "If implementation `B` is not secure, then implementation `A` is not secure."

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.

Task 4 – Would You Like Implies With That? [10 pts]

You are presented with four *two-sided* (one green, one white) cards:



On the green side of each card is a letter, and on the white side is a number.
Consider the following rule:

> If a card has a vowel on one side, then it has an odd number on the other side.

Which card(s) *must* be turned over to check if the rule is true? Explain your answer in a few sentences.

Task 5 – Super-Low Mortgage Gates [20 pts]

For this problem, we have invented a new gate, called an "A" gate, which is defined as follows:

r	s	t	$A(r,s,t)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

Note that this gate has *three inputs*, rather than just two (like `AND` and `OR`) or one (like `NOT`). Also, note that we are using function syntax for an "A" gate. The same could be done for `AND` and `OR` gates. For example, we could write $\texttt{AND}(r,s)$ rather than $r \wedge s$ and $\texttt{OR}(r,s)$ rather than $r \vee s$.

Demonstrate that we can construct each of the operators below. For each one, first give a formula that is equivalent to that operator but *that uses only As and no other gates*. Second, justify your answer with a truth table containing columns for both the original operator and your formula. If you are using multiple A gates, the output of each one should have its own column in the table.

Hint: It is okay to use the same propositional variable more than once. It is also okay to hard-code inputs of an "A" gate to be a constant 0 or 1.

a) $\neg p$ using *only one* A gate

The answer should be of the form $A(r,s,t)$, where each of r, s, t is replaced by any of p, 0, 1.

b) $p \vee q$ using *only one* A gate

c) $p \rightarrow q$ using *only one* A gate

d) $q \rightarrow \neg p$ using *at most two* A gates

The answer should still be of the form $A(r,s,t)$, but now, one of r, s, t can itself be of the form $A(u,v,w)$, where each of u, v, w is any of p, q, 0, 1.

e) $p \wedge q$ using *at most three* A gates

Hint: Look closely at the truth table for $q \rightarrow \neg p$ and see if you can spot a relationship with $p \wedge q$.

Task 6 – Kellogg's Form Flakes [14 pts]

The Java project you are working on contains the following function. It takes four boolean arguments, a, b, c, and d, and returns true if at least two of them are true or if b, c, and d are all false:

```
public static boolean E(boolean a, boolean b, boolean c, boolean d) {
  if (!b && !c && !d) {
    return true;
  } else {
    int count = 0;
    if (a) count += 1;
    if (b) count += 1;
    if (c) count += 1;
    if (d) count += 1;
    return 2 <= count;
  }
}
```

In this problem, you will write a simpler, faster version of this function.

a) Write a truth table for E. Include columns showing whether at least two of the variables are true and whether b, c, and d are all false before the final column showing the value of $E(a, b, c, d)$.

b) Write a Boolean algebra expression for E in sum-of-products form.

c) Write a Boolean algebra expression for E in products-of-sums form.

d) Fill in the body of `E` with a Java expression that returns the same value as the original code.

```
public static boolean E(boolean a, boolean b, boolean c, boolean d) {
  return

}
```

Task 7 – Extra Credit: XNORing

Imagine a computer with a fixed amount of memory. We give names, $R_1, R_2, R_3, \ldots$, 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 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).