CSE 311: Foundations of Computing I Spring 2025

Problem Set 1

Due: Wednesday, Apr 9th 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:

- Each numbered task should be solved on its own page (or pages). Do not write your name on the individual pages. (Gradescope will handle that.)
- When you upload your pages, make sure each one is **properly rotated**. If not, you can use the Gradescope controls to turn them to the proper orientation.
- Follow the Gradescope prompt to **link tasks to pages**.
- 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.

Task 1 – Stop, Prop, and Roll [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 I haven't had my coffee and the sun is not up, then I am angry. But if I have had my coffee or the sun is up, then I am happy."

b) Define a set of *at most four* atomic propositions. Then, use those propositions 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 pop and push.

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

i) You can have your cake or you can eat your cake, but not both.

ii) If you have your cake and you drop your cake, then you are sad and you don't have your cake. But if you eat your cake and are sad, then you don't have your cake.

Task 2 – Good to the Last Prop [12 pts]

Consider this sentence: Andre goes out with friends if it's a Friday or if Andre doesn't have both homework and an exam on the next day.

a) Define a set of *four* 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 – I've Got Friends in High Cases [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** "If this implementation (`A`) is secure, then implementation `C` is secure."
>
> **`B`'s documentation** "If implementation `B` is not secure, then implementation `C` is not secure."
>
> **`C`'s documentation** "This implementation (`C`) is not secure."

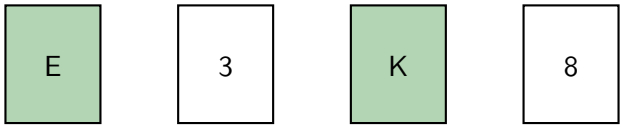
Unfortunately, the documentation in the repo also tells you that **only one documentation sentence is true** — the other two are false!

Which implementation is the *secure* one? (Note that it is not necessarily the one whose documentation sentence is true!)

Justify your answer with a **case analysis**. I.e., consider each possibility for which implementation is the secure one and show which of the sentences would be true in each case. Only one case should match the description above.

Task 4 – Give Me Some of the Card Stuff [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 an even number on the one side, then it has a vowel 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 – The Calm Before the Form [14 pts]

The Java project you are working on contains the following function. It takes four boolean arguments, a, b, c, and d, and calculates some boolean function of them called E:

```
public static boolean E(boolean a, boolean b, boolean c, boolean d) {
  if (!(a || b))
    return false;
  if (!(!a || !b))
    return false;
  if (!(a || c))
    return false;
  return (b || !d);
}
```

The code checks the value of the four disjunctions $a \vee b$, $\neg a \vee \neg b$, $a \vee c$, and $b \vee \neg d$ and returns true iff *all four* of them are true. In other words, it calculates the conjunction (and) of those four expressions. Specifically, it calculates the same value as the following (non-canonical) CNF expression:

$$(a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee c) \wedge (b \vee \neg d)$$

Note that this Java code could be written equivalently as a single `return` statement:

```
return (a || b) && (!a || !b) && (a || c) && (b || !d);
```

The Java compiler would automatically translate that into the series of `if` statements above that returns `false` as soon as one disjunction is found to evaluate as false, a process known as "short circuiting".

a) Write a truth table for E. Include columns for a, b, c, d, all four disjunctions, and E.

b) Write the **canonical** DNF expression for E.

c) Translate your DNF expression into a new Java implementation of `E`.

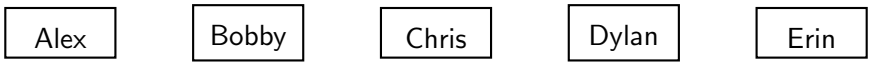
Like above, your code should be a series of `if` statements and then a `return`, except that, now, each `if` should check the value of a conjunction and return `true` if it is true.

(Do not simplify. Translate your expression to code in the most direct way possible.)

Task 6 – Let the SAT Out of the Bag [22 pts]

You are organizing a holiday party for five friends. Each of them needs a party hat, which are available in 3 colors: red, green, and blue. It is up to you to choose the color of each friends' hat.

In addition, there are some rules about how hat colors must be chosen. First, no one should have the same color hat as a person seated next to them. Here is the seating chart:



In addition, each of these pairs must not have the same hat color:

- Alex and Dylan: they are mortal enemies
- Bobby and Dylan: it annoys them when they wear the same color and people think they're dating
- Bobby and Erin: they are secretly dating and want throw everyone off the scent

There are $3^5 = 243$ different choices of hat colors, so we don't want to try them all! Instead, we will use a SAT solver.

For each of the five people, we will have 3 boolean variables. For Alex, let's call these a_R, a_G, and a_B. The variable a_R indicates that Alex's hat is red, while a_G and a_B indicate that it is green and blue, respectively. Of course, these variables cannot be assigned arbitrarily. For Alex, we need to make sure that exactly one of the three variables is true. Let's start by figuring out how to do that...

a) Write a truth table in the three variables a_R, a_G, and a_B that shows when exactly one of them is true. (Your table should have four columns and 8 rows.)

b) Write the canonical CNF formula for the boolean function in the last column.

By repeating the formula above also for Bobby, Chris, Dylan, and Erin's three variables, we can ensure that we only have assignments that pick a single hat color for each person. The full CNF formula (so far) would have a total of 25 conjuncts: 5 conjuncts times 5 people.

We also need to ensure that, for each of the 7 pairs mentioned above, the same hat color is not selected. For example, consider the pair of Alex and Dylan and call Dylan's variables d_R, d_G, and d_B. We need to make sure that we don't have a_R and d_R both true. Let's look at how to do that next...

c) Write a truth table in the two variables a_R and d_R that is false only when both are true.

d) Write the canonical CNF formula for the boolean function in the last column.

e) Write a CNF formula that is true when Alex and Dylan do not have the same hat color. (Your answer should have only 3 conjuncts.)

By repeating the formula above for each of the 7 pairs, we can ensure that no pair has the same color. This requires a total of 21 conjuncts: 3 per pair times 7 pairs.

All together, our hat assignment problem uses 15 variables and $21 + 25 = 46$ conjuncts to describe a valid choice of hat colors. That is only a fraction of the 243 combinations we would need for a truth table. This difference only expands as we add more people. With 15 people, the SAT problem would have only 45 variables and, at most, 390 conjuncts, while the truth table would have over *14 million* rows! However, to make this problem easier, we've stuck to 5 people.

f) Use the SAT solver on this page to solve the problem:

https://homes.cs.washington.edu/~kevinz/sat-solver/

It has instructions for how to write your CNF formula in the DIMACs format required by the tool.

In your solution, include the input you gave to the tool and its output. Then, describe what the output means: which assignment of colors did it pick?

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).