Problem Set 1

Due: Wednesday, April 5, by 11:59pm

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 homeworks 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 **four** late days during the quarter, but can only use up to two late days on any one problem set. Please plan ahead, as we will not be willing to add any additional late days except in absolute, verifiable emergencies. The final problem set will not be accepted late.

Solutions submission. You must submit your solution via Gradescope. In particular:

- Submit a *single* 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 prompt on Gradescope to link tasks to your pages.
- Do not write your name on the individual pages – Gradescope will handle that.
- We encourage you to typeset your solution. The homepage provides links to resources to help you doing so using LaTeX. If you do use another tool (e.g., Microsoft Word), we request that you use a proper equation editor to display math (MS Word has one). For example, you should be able to write $\sum_{i=1}^{n} x^i$ instead of `x^1 + x^2 + ... + x^n`. You can also provide a handwritten solution, as long as it is on a single PDF file that satisfies the above submission format requirements. It is your responsibility to make sure handwritten solutions are readable – we will *not* grade unreadable write-ups.

Task 1 – Watch Your Language [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 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) 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) 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`.

Task 2 – Doublespeak [12 pts]

Consider the following sentence: If we have lecture today and Paul is out of town, James 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, simplify it so that all $\neg$ symbols are next to atomic propositions.

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 – Same Difference [16 pts]

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 must include columns for all subformulas.

a) $(P \wedge Q) \wedge (P \vee Q)$ vs. $P \vee Q$

b) $P \oplus Q$ vs. $\neg P \oplus \neg Q$

c) $P \rightarrow (Q \rightarrow R)$ vs. $(P \wedge Q) \rightarrow R$

d) $(P \rightarrow Q) \rightarrow R$ vs. $(Q \rightarrow P) \vee R$

Task 4 – Same to You [16 pts]

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. (Our 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) $P \to (Q \to R) \equiv (P \land Q) \to R$

b) $(\neg P \to Q) \land (Q \to P) \equiv P$

c) $(P \to Q) \lor (P \to \neg Q) \equiv \mathsf{T}$

d) $((P \lor \neg P) \to P) \land (Q \to P) \equiv P$

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 Cozy homepage, at the the link labelled "Docs" at the top of the page.

Task 5 – Case Closed [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."

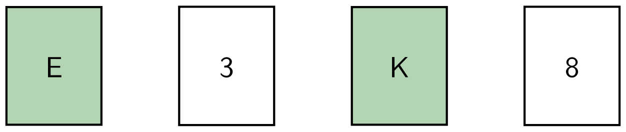
`C`'s documentation "If implementation `B` is not secure, then implementation `A` is not secure."

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 6 – Card Tricks [8 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 even 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 7 – All You Need is $\divideontimes$ [20 pts]

For this problem, we have invented a new operator called $\divideontimes$. It is defined by the following truth table:

p	q	$p \divideontimes q$
T	T	F
T	F	T
F	T	T
F	F	T

Demonstrate that we can construct each of the operators below as follows. For each operator below, first give a formula that is equivalent to that operator but *that uses only $\divideontimes$ and no other operators and no constants like T or F*. Second, justify your answer with a truth table containing columns for both the original operator and your formula. Your truth table must also include columns for all subformulas of your formula, as usual.

Hint: It's okay to use the same propositional variable more than once.

a) $\neg p$

b) $p \lor q$

c) $p \land q$

Task 8 – Extra Credit: XNORing

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).