Homework 1 (due April 08 2020)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough to convince someone who does not already know the answer. You may use results from lecture and previous homeworks without proof. See the syllabus for more details and for permitted resources and collaboration.*

1. Translation to logic (25 points)

Translate these English statements into logic, making the atomic propositions as basic as possible.

(a) [5 points] The file can be created only if the disk is not full and the user has write permissions.

(b) [10 points] Define a set of *at most three* atomic propositions. Then, use them to translate all of these sentences about what analyses can be used in a compiler:

1. If the analysis is sound, then it can be used in a compiler.
2. Unless it is fast, the analysis cannot be used in a compiler.
3. The analysis can be used in a compiler only if it is sound and fast.

(c) [10 points] Define a set of *at most four* atomic propositions. Then, use them to translate all of these sentences about how a bounded queue data structure can be used into logic:

1. You can enqueue to but not dequeue from the queue if it is empty.
2. Only if the queue is full can you dequeue from but not enqueue to it.
3. The queue is neither empty nor full unless you cannot dequeue from it or cannot enqueue to it.

2. Inequivalent logical statements (20 points)

For each part, find truth values for the propositional variables p, q, and r to show that the pair of statements are inequivalent. Explain why your answers work.

(a) [5 points] $p \wedge (p \vee q)$ vs. $p \wedge q$

(b) [5 points] $\neg(p \oplus q)$ vs. $\neg p \oplus \neg q$

(c) [5 points] $(p \to q) \to r$ vs. $q \to (r \to p)$

(d) [5 points] $(p \to q) \to (q \to p)$ vs. $(\neg q \to p) \to (p \to \neg q)$

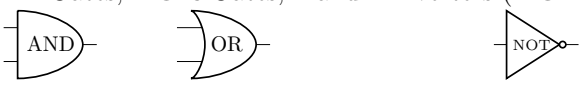
3. Finding compound propositions (10 points)

Find a compound proposition involving the propositional variables p, q, and r that is true precisely when a minority of p, q, and r are true. Explain why your answer works.

4. Drawing circuits (15 points)

Consider the following gates:

AND Gates, **OR** Gates, and Inverters (**NOT** Gates).



Using only **AND**, **OR**, and **NOT** gates, draw the diagram of a circuit with **three inputs** that computes the function $B(p, q, r)$, defined as follows:

- If both p and q are true, then $B(p, q, r) = \neg r$.
- If exactly one of p and q is true, then $B(p, q, r) = r$.
- Otherwise, we have $B(p, q, r) = \mathsf{F}$.

5. Implementing gates (20 points)

Define the "U" gate by the rule $U(s, r) := r \vee \neg s$. Show how to implement the following gates using only U's. You are allowed to use the constants `T` and `F` and the inputs p and q. You can use inputs multiple times. You must *justify your answers.*

(a) [5 points] $\neg p$, using only **one** U gate.

(b) [5 points] $p \vee q$, using at most **two** U gates.

(c) [5 points] $\neg(p \wedge q)$, using at most **two** U gates.

(d) [5 points] $p \wedge q$, using at most **three** U gates.

6. Puzzling documentation (10 points)

Searching for a secure encryption procedure, you find a GitHub repo with three implementations. Only one of them is secure; the other two are not. All three implementations are obfuscated, so you cannot tell which one is secure by examining the source. But each implementation has one documentation sentence:

`A` "This implementation is not secure."

`B` "This implementation is not secure."

`C` "Implementation `B` is secure."

Only one documentation sentence is true; the other two are false. Which implementation is secure? Justify your answer.

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 another 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 p and R_2 containing the value of the proposition q, 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 $\neg p \vee (p \wedge q)$. Note that this last instruction reads from R_4 and also stores the result into R_4. This is allowed.

Now, assuming p is stored in register R_1 and q 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 q stored in R_1 and p stored in R_2 (i.e., with the original values in R_1 and R_2 swapped).