CPSC 121 Quiz 1
Wednesday, 2012 July 4

[2] 1. Here's a short version of a statement from the UBC calendar: "You must meet the promo reqts. If you do not meet them, you do not get to stay in the Faculty."[1]

Let m mean "I met the promo reqts." Let s mean "I stay in the Faculty."

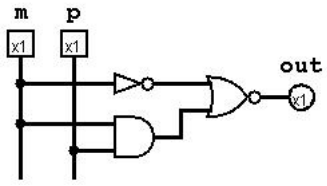
Give a propositional logic statement with the same meaning as the statement above. (Be careful; a direct translation probably isn't useful. Instead, write a statement with the same truth table as the English statement.)

Solution : There are four possible truth values for the pair of variables m and s. Clearly if you don't meet the reqts and yet you stay in the Faculty, that violates the statement. So, it's false then. Pretty clearly it's true if you do meet the requirements and you do stay or you don't meet the requirements and you don't stay. How about if you meet the requirements and don't stay? Well, maybe you transferred to Hogwarts. The statement is still true.

So, we get something like $\sim m \rightarrow \sim s$ or $m \vee \sim s$.

[4] 2. Write a circuit that corresponds *directly* to $\sim(\sim m \vee (m \wedge p))$.

Solution :



(As always, the NOR gate could be an OR followed by a NOT instead.)

[3] 3. Write out a truth table for the statement $\sim(\sim m \vee (m \wedge p))$.

Solution : Note: Writing out intermediate columns was worth partial credit. It's worth doing if you make mistakes jumping to the last column.

m	p	$\sim m$	$m \wedge p$	$\sim m \vee (m \wedge p)$	$\sim(\sim m \vee (m \wedge p))$
T	T	F	T	T	F
T	F	F	F	F	T
F	T	T	F	T	F
F	F	T	F	T	F

[2] 4. Actual gates can only send their output along to a limited number of inputs of subsequent gates. Indicate at least one way that the propositional logic model falls short with respect to this issue: failing to capture useful information that *is* present in a circuit diagram.

[1] NOT RELEVANT TO THE QUIZ. Here is the full version (lightly edited): "Students must meet the second-year promotion requirements within 48 attempted credits after admission to first year. Those who do not will be required to withdraw from the Faculty."

Solution : In a circuit diagram, we can trace from the output of a gate to the subsequent circuit elements it "drives" (inputs of gates after it). In a propositional logic statement, there is no direct representation of the number of connections to subsequent gates. The best we can probably do is count repeated subexpressions to estimate how many subsequent gates some gates *might* connect to, depending on wiring choices.

[9] 5. Write out propositional logic statement(s) that could be directly translated into a circuit for this problem: given a 3-bit number as input, produce the largest factor of the number other than itself as output. As special cases, 0 and 1 should both produce 0. The output should be a 2-bit number. (So, for example, 7 produces 1 because it has only 1 and itself as factors. 6 produces 3 because it has 1, 2, 3, and itself as factors.)

Reminders: For a 3-bit number, 000 = 0, 001 = 1, 010 = 2, 011 = 3, 100 = 4, 101 = 5, 110 = 6, and 111 = 7. For a 2-bit number, 00 = 0, 01 = 1, 10 = 2, and 11 = 3. A factor of a number is a positive integer that divides that number.

CLEARLY SHOW YOUR WORK for partial credit.

Solution : Here's a truth table with extra columns showing the input and output numbers in decimal. Only 2, 3, 4, and 5 remained to solve. 2, 3, and 5 have no factors except 1 and themselves. (They're prime.) So, the output is 1. 4 has 2 as a factor.

i	i_1	i_2	i_3	o_1	o_2	o
7	T	T	T	F	T	1
6	T	T	F	T	T	3
5	T	F	T	F	T	1
4	T	F	F	T	F	2
3	F	T	T	F	T	1
2	F	T	F	F	T	1
1	F	F	T	F	F	0
0	F	F	F	F	F	0

Since there are two bits of output, we need two propositional logic statements.

We'll do the mechanical solution, although we model the F rows and not the T rows for o_2.

$o_1 = (i_1 \wedge \sim i_2 \wedge \sim i_3) \vee (i_1 \wedge i_2 \wedge \sim i_3)$.

$o_2 = \sim[(i_1 \wedge \sim i_2 \wedge \sim i_3) \vee (\sim i_1 \wedge \sim i_2 \wedge i_3) \vee (\sim i_1 \wedge \sim i_2 \wedge \sim i_3)]$

BONUS: Earn up to 2 bonus points by doing one or more of these problems.

- Draw the simplest possible circuit to solve the last problem above. (Otherwise, you don't need to draw a circuit for that problem at all.) Hint: simplify your propositional logic first!

Solution : We don't provide a full solution, but note: whenever i_2 is true, so is o_2. Also note that we can use distributivity on the basic expression we'd get with our mechanical technique for o_1: $(i_1 \wedge \sim i_2 \wedge \sim i_3) \vee (i_1 \wedge i_2 \wedge \sim i_3)$. (Neither of these is a stopping point; they're starting points!)

- There's a gate called a "buffer" with the truth table:

in	out
T	T
F	F

This seems useless, but something on this quiz gives one of the justifications for the gate. Give a clear example of how and why we might use the buffer, including how best to use it when the problem it solves is severe.

Solution : Imagine we use the output of a gate too many times in a circuit. We could "split" the gate in two (calculating its value twice) and thus reduce the number of times each gate is used (assigning half of the subsequent gates to one of the new gates and half to the other).

However, the same problem may actually happen for an input. We can't just decide to split the gate that created that input because it's not available.

(We could try to go and grab the gate that provides that input. Two problems with that. First, our circuit may be used in multiple places, and we cannot necessarily predict where. Second, even if it weren't, we're making *big* changes to the circuit to solve a *local* problem. That sort of change almost always an error in Computer Science. It violates the notions of "abstraction", "encapsulation", and "modularization", aptly illustrated by the UW CSE band's "Ease of Change" in the phrase "Hide what you have to hide / so you can change what you have to change." `http://www.cs.washington.edu/orgs/student-affairs/cseband/`)

So, what do we do instead? We can feed our input into a buffer and feed the output of the buffer to multiple circuit elements. Even if we exceed the capacity of the buffer, we can just send the buffer's output to as many buffers as it can drive and then send those buffers' outputs along to the rest of the circuit, repeating this process as necessary. (This is the same overarching idea that you'll see in the scalability lab.)