Problem Set #1

Warning: You have already made the maximum number of submissions. Additional submissions will not count for credit. You are welcome to try it as a learning exercise.

Question 1

Let Boolean function $F(x, y, z, w) = (xy + \bar{x}z) \oplus w$. Which of the following equations involving Shannon Cofactors are correct?

Select all the correct solutions. There may be more than one correct solution.

(Reminder: $a \oplus b = a\bar{b} + \bar{a}b$; think carefully about what equations like (Boolean stuff)$\oplus$1 and (Boolean stuff)$\oplus$0 can simplify to.)

- $F_{\bar{x}\bar{y}} = w$
- $F_{xy} = \bar{w}$
- $F_w = \overline{xy + \bar{x}z}$
- $F_x = y \oplus w$
- $F_{\bar{x}} = \bar{z} \oplus w$

Question 2

There are other ways of representing the Shannon Expansion theorem. The version we gave you – $F(x_1, \ldots x_i, \ldots x_n) = x_i \bullet F(x_i = 1) + \bar{x_i} \bullet F(x_i = 0)$ can be thought of as a “sum of products” form, since the equation is an OR (sum) of two small AND (product) terms. But there must be a "product of sums" form for the Shannon expansion. Use Boolean algebra, and the basic properties of cofactors, and tell us which of these equations is a correct alternate form of the Shannon expansion.

There may be more than one correct answer; if so, select each correct answer.

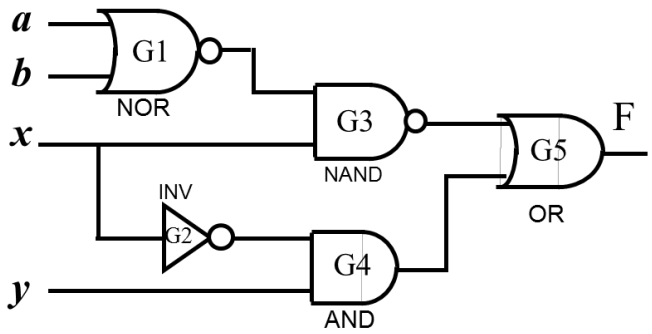
- $[x_i + F(x_i = 1)] \bullet [\bar{x}_i + F(x_i = 0)]$
- $[x_i + F(x_i = 1) + \bar{x}_i + F(x_i = 0)]$
- $[x_i + \bar{x}_i] \bullet [F(x_i = 0) + F(x_i = 1)]$
- $[x_i + F(x_i = 0)] \bullet [\bar{x}_i + F(x_i = 1)]$

Question 3

Consider the small logic network shown below. Use the definition of the Boolean Difference and determine which of these equations is correct.

There may be more than one correct answer; select all correct equations.

(Reminder: $p \oplus q = p\bar{q} + \bar{p}q$. Think: $p \oplus p$ = what? $p \oplus \bar{p}$ = what?)



- $\frac{\partial f}{\partial x} = \bar{a} \cdot \bar{b}$
- $\frac{\partial f}{\partial x} = \overline{ab}$
- $\frac{\partial f}{\partial y} = 1$
- $\frac{\partial f}{\partial y} = \bar{a} + b + \bar{x}$
- $\frac{\partial f}{\partial y} = 0$

Question 4

Looking at the same gate network from Problem 3, which are correct Boolean formulas for these quantification operations on F?

- $(\exists x, f)[a, b, y] = 0$
- $(\forall x, f)[a, b, y] = (\bar{a} + \bar{b}) \cdot y$
- $(\forall x, f)[a, b, y] = a + b$
- $(\exists x, f)[a, b, y] = \bar{a} + \bar{b} + \bar{y}$
- $(\exists x, f)[a, b, y] = 1$

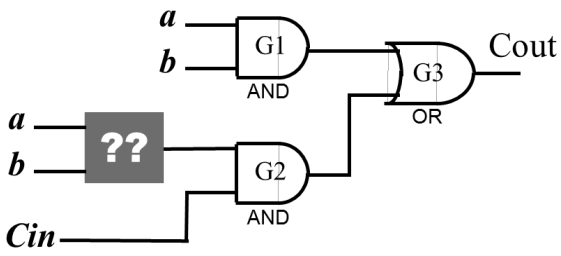
Question 5

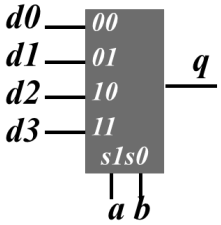
Network repair

The carry output of a 1-bit adder has the Boolean equation $Cout = a \cdot b + (a + b) \cdot Cin$, where a and b are the 1-bit numbers we want to add, and Cin is the input carry bit. Suppose we know that we have implemented the gate-level logic incorrectly, the gate with the "??" label is the one we suspect is incorrect.

Do this: use the logic network repair procedure discussed in lecture; replace the "??" gate with the 4:1 multiplexor shown below, with inputs a,b connected to the $s1$,$s0$ select lines. (So, for example, $s1s0 = 10$ means $q = d2$ for this mux.) Use the quantification procedure, and do the Boolean algebra to see what the result tells you about how to fix it.

Select the right answer below.



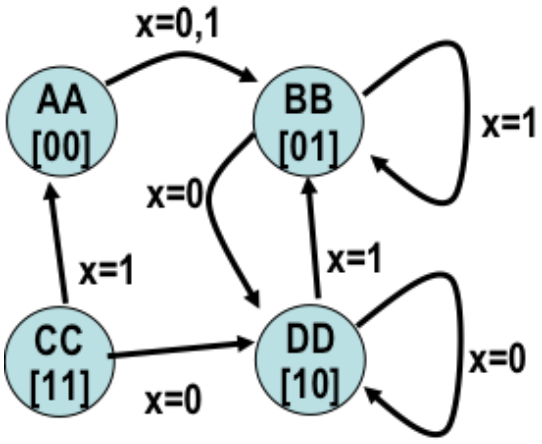


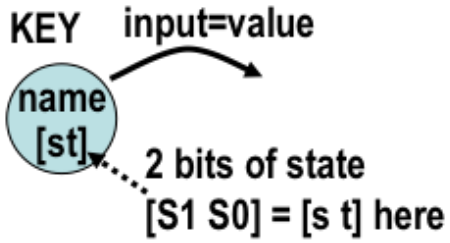
- There are two feasible repairs: replace “??” with either an OR or an EXOR gate.
- It’s a trick! There is NO feasible repair of this network!
- The only feasible repair is to replace the “??” with a NOR gate.
- The only feasible repair is replace the “??” with an EXOR gate.
- The only feasible repair is to replace the “??” with an OR gate.

Question 6

Using quantification for finite state machine (FSM) analysis.

It is very natural to think about representing a finite state machine using some kind of a graph: the *nodes* represent the states, and the *edges* represent the transitions from state to state under input changes. The common state diagram notation (see diagram) is exactly this kind of a graph:





This machine has four states AA BB CC DD, represented by 2 bits of state (2 flip flops) whose values appear in brackets (e.g., [11] represents state CC) in each state bubble. This machine has just 1 input x. We are ignoring any outputs for

this machine.

Unfortunately, these graphs can get very big, very easily. Imagine a state machine with 20 bits of state (20 separate flip flops). It has 2^{20} states $\sim$ 1,000,000 states. Suppose also there are 20 input variables. Then each of these 1,000,000 states has 1,000,000 transition arrows leaving it. Our little machine with 20 flip flops has one trillion state transitions!

It turns out there is a more elegant way of representing things here. Imagine that we create a **new** Boolean function, called **R**, which we call the *state transition relation function*. In general **R** has 3 kinds of inputs:

> **R**(*state variables for starting state*, *input variables*, *state variables for ending state*)

The way to think about the **R** function is that it answers a simple question: *can I get from a specific starting state to a specified ending state via specified input value*? For our little example, here are a few values of **R**:

> **R**(AA, 0, BB) = 1 means "*Yes, from state AA, an input x=0 takes you to state BB*"
> **R**(BB, 1, AA) = 0 means "*No, from state BB, input x=1 does NOT take you to AA*"

Of course, you can't just input the state "names", you have to use the state assignment bits that represent each state. This means that **R** is a function of 5 variables for our little example:

> **R**(S1, S0, x, E1,E0),

where *S1 S0* is the state assignment for the starting state, and *E1 E0* is the assignment for the ending state. So, for example:

> **R**(AA, 0, BB) = 1 really means that R(0, 0, 0, 0, 1) = 1, since state AA is represented by the assignment S1 S0 = 00, and similarly the state BB has the assignment 01.

You could easily build a truth table for **R**( ) for this little 4-state example, and

create a sum of products Boolean expression for **R**(S1, S0, x, E1, E0) using a 5-variable Kmap, or just ordinary Boolean algebra. All the information you need to build **R**( ) is available in our state machine design.

Now, let's create a more interesting function, called **G**(*E1*, *E0*). **G** is again a function of the states, but it answers a different question: *is there ANY way to reach this state E1 E0*? For example, it is clear from our little state diagram that **G**(CC) = 0, i.e., there is just NO way to start from some state and take a transition that gets you to state CC. But **G**(AA)=1 because it *is* possible to get to state AA (for example, by being in state CC and taking the x=1 transition).

For a trivial example like our little 4-state machine, you can just look at the right answer. For a real machine, with perhaps billions of states+edges, you cannot just look at the graph. So, answer this: Can we use computational Boolean algebra and transform the **R( )** function – which we can build – in the **G( )** function, directly?

Select the correct answers below.

- Yes, we can compute this as $(\forall S1S0x, R)[E1, E0]$, because we need to know "for all" end states (E1,E0) how the start state (S1,S0) and input transition allow us to reach this end state.
- Yes, we can compute this as $\frac{\partial R}{\partial S1 \partial S0 \partial x}[E1, E0]$, since we want to know the sensitivity of the end state E1,E0 to the start state S1S0 and input x.
- Yes, we can compute this as $(\exists S1S0x, R)[E1, E0]$ because all we really need to know is that "there exists" some start state (S1S0) and some transition (x) that does lead to this end state (E1E0), and we don't need to know what those are, just that "they exist".
- No, it's a trick! There is no way to use the computational Boolean algebra methods we just learned to do this!

Question 7

Unate functions

Consider this Boolean function $F(x, y, z, w) = (xy\bar{w} + \bar{y}\,\bar{z} + xz\bar{w} + yz\bar{w})$

Which of these are true? Select all correct answers.

- [ ] F is positive unate in variables (x, y) and negative unate in variables (w, z)
- [ ] F is unate in all variables
- [ ] F is binate in variables (y, z) and unate in all other variables
- [ ] F is positive unate in variable (y) and negative unate in variable (z)
- [ ] F is positive unate in variable (x) and negative unate in variable (w)

Question 8

This question has no correct answer

See this thread on the forum for more context.

Thus, we are going to nullify this question. *Don't worry*: if you have submitted your quiz before (with **whatever** answer choice), you will receive full points for this part. You might not see the updated score right away but we will issue a regrade and all will be well again.

What you should do, in place of this problem, is take Problem Set #1a that will have a **fixed** version of this problem. We are leaving the question as-is here just for your reference.

Positional Cube Notation and manipulations.

Consider this Boolean equation $F(x, y, z, w) = xy\bar{z}\,\bar{w} + y + \bar{x}\bar{y}\,\bar{z}\,w + x\bar{z}\,\bar{w}$

If we transform this into PCN form, where each cube is [xx yy zz ww], a set of 2-bit entries for each variable in order, it will become the following list:

[01 01 10 10], [11 01 11 11], [10 10 10 01], [01 11 10 10]

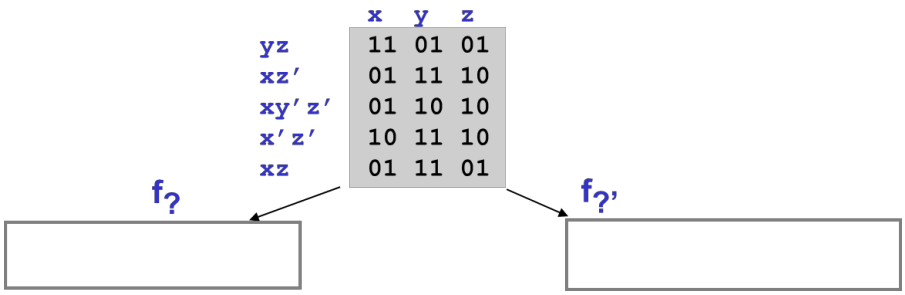
Which of the following is true, starting with this PCN list, using the methods and rules from URP tautology lecture?

- [ ] F($w = 1$) = [11 01 11 01], [10 10 10 11] and this cofactor is binate.
- [ ] F($z = 1$) = [11 01 01 11]
- [ ] F($x = 0$) = [10 01 11 11], [11 10 10 01] and this cofactor is unate.
- [ ] F($y = 1$) = [01 11 10 10], [11 11 11 11], [01 01 10 10] and this cofactor is unate.
- [ ] F($w = 0$) = [11 01 11 10], [10 10 10 11]
- [ ] F($y = 0$) = [01 11 10 10], [11 11 11 11], [10 10 10 01], [01 11 10 10]
- [ ] F($z = 0$) = [01 01 11 10], [11 01 10 11], [10 10 11 01], [01 11 11 10] and this cofactor is unate.
- [ ] F($x = 1$) = [11 01 10 10], [01 01 11 11], [11 11 10 10]

Question 9

Unate Recursive Paradigm algorithm

Suppose we have these cubes at one node of our URP tautology recursion, and we need to decide on the splitting variable to use to cofactor and recurse. Which variable do we pick, and why? Select the right answer.

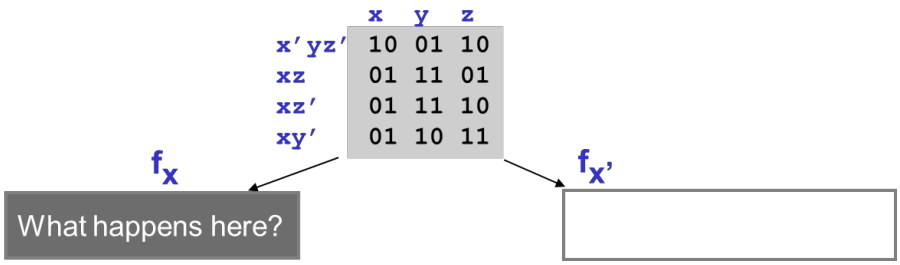


- Variable y is the most binate variable since it appears in 5 cubes.
- Variable x is the only binate variable, and so we select it.
- Variable z is the most binate variable since it appears in 5 cubes.
- Variables y and z are tied for most-binate, since they each appear in 5 cubes, but we select y because it has the smallest value of $|$True cubes – Complement cubes$|$. This value is 0 for variable y and 1 for variable z, so we pick y.

Question 10

Unate Recursive Paradigm.

Consider this new cube list over variables x, y, z, which appears at one node of our URP tautology algorithm. We select variable x to split on. Let us consider what happens when we recurse, and call Tautology on the positive x cofactor.



Which of the following are true statements for the cubelist that should appear inside the f(x=1) recursive call, highlighted in our figure?

- The cube list is [01 11 01], [01 11 10], [01 10 11] and it is not a tautology.
- The cube list is [11 11 01], [11 11 10], [11 10 11] and it is a tautology
- The cube list is [01 11 01], [01 11 10], [01 10 11] and we cannot tell if it is tautology, we must recurse.
- The cube list is [11 11 01], [11 11 10], [11 10 11] and it is not a tautology
- The cube list is [11 11 01], [11 11 10], [11 10 11] and we cannot tell if it is a tautology and we must recurse some more

☐ In accordance with the Honor Code, I certify that my answers here are my own work.

Submit Answers Save Answers