Written Homework 02

Assigned:  Th 25 Jan 2018
Due:  Wed 31 Jan 2018

Instructions:

• The assignment has to be uploaded to Blackboard by the due date. NO assignment will be accepted after 11:59pm on that day.

• We expect that you will study with friends and often work out problem solutions together, but you must write up your own solutions, in your own words. Cheating will not be tolerated. Professors, TAs, and peer tutors will be available to answer questions but will not do your homework for you. One of our course goals is to teach you how to think on your own.

• You may turn in work to Blackboard that is either handwritten and scanned, written in a word processor such as Word, or typeset in LaTeX. In the case of handwritten work, we may deduct points if the scan is upside down or the work is illegible.

• To get full credit, show INTERMEDIATE steps leading to your answers, throughout.

• When asked for a logical formula, use ∧, ∨, and ¬ for AND, OR, and NOT.

Problem 1  [25 pts (5 points each)]: Logical Equivalence

Two programmers are arguing about whether a rewritten condition is equivalent to the original. The first programmer originally wrote the line

if ((not (a and b)) or ((not a) or (not b)))

and the second programmer wanted to simplify this to

if ((not a) or (not b)).

i. Make a truth table that contains columns for both the longer condition and the shorter one.

Solution:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>¬(a ∧ b) ∨ (¬a ∨ ¬b)</th>
<th>(¬a ∨ ¬b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ii. Show step-by-step that the first condition simplifies to the second. Name the logical rules that you are invoking as you simplify. (The rules can be found in Table 3.4 of the first edition of the textbook.)

Solution: ¬(a ∧ b) ∨ (¬a ∨ ¬b)

becomes (¬a ∨ ¬b) ∨ (¬a ∨ ¬b) (De Morgan’s Law)
becomes \((\neg a \lor \neg b)\) (Idempotence law)

The idempotence law is here being applied to a whole repeated clause, but we could instead apply it to each pair of terms after getting rid of the parentheses (associative law) and rearranging the order (commutative law).

**iii.** As it so happens, a third programmer comes along and berates both programmers for using more operations than necessary. How could the original condition be written with just two AND, OR, and/or NOT operators?

*Solution:* It can be written as \((\neg (a \land b))\)

**iv.** The programmers were programming a simulated robot. When the robot is actually built, the electrical engineer building it ignores the programmers’ coding choices and implements this condition with a single logic gate that is equivalent. Draw it.

![Logic gate diagram](image)

*Solution:*

**v.** Elsewhere in the code base, an intern wrote an even larger conditional, equivalent to the Boolean formula \(((\neg a \land b) \lor (a \land b)) \lor (b \land \neg a) \lor a\). Simplify this formula. You don’t need to name logical rules for each step.

*Solution:* \(a \lor b\). \(((\neg a \land b) \lor (a \land b))\) simplifies to \(b\), and \(b \lor (b \land \neg a)\) also simplifies to \(b\).

---

**Problem 2** [25 pts (6,7,6,6)]: DNF, CNF, and 2 of 4

A programmer wants to write a conditional statement that is true if exactly 2 of 4 conditions A, B, C, and D are true – no more, and no less.

**i.** Write a DNF formula that is true when exactly two of the four inputs A, B, C, D are true.

*Solution:* \((A \land B \land \neg C \land \neg D) \lor (A \land \neg B \land C \land \neg D) \lor (A \land \neg B \land \neg C \land D) \lor (\neg A \land B \land C \land \neg D) \lor (\neg A \land B \land \neg C \land D) \lor (\neg A \land \neg B \land C \land D)\)

**ii.** In a different place, the programmer needs to write a condition that is true unless exactly three of the conditions A, B, C, or D are true; if exactly three inputs are true, the condition should evaluate to false. Write this formula in CNF. (You can do this by writing one clause per line of the truth table that is false; for each such row, write a clause that is true exactly when the variables don’t match that row.)

*Solution:* \((\neg A \lor \neg B \lor \neg C \lor D) \land (\neg A \lor \neg B \lor C \lor \neg D) \land (\neg A \lor B \lor \neg C \lor \neg D) \land (A \lor \neg B \lor \neg C \lor \neg D)\)
iii. In yet a different place in the code, the condition should output true if at least 2 of 4 inputs are true. Write this new condition in DNF without using any NOT operators.

Solution: \((A \land B) \lor (A \land C) \lor (A \land D) \lor (B \land C) \lor (B \land D) \lor (C \land D)\)

iv. Draw a circuit based on your answer to the previous question that has two outputs: one that is true if at least 2 of 4 inputs A,B,C,D are true; and a second output that is true only if strictly fewer than 2 inputs are true. You should achieve the second effect using as few gates as possible.

Solution:

Problem 3 [10 pts]: Building a multiplication circuit

i. Draw a circuit composed of logic gates and half-adders that takes two unsigned 2-bit binary numbers as input, and produces the four-bit result of multiplying those numbers as output. Label your inputs \(a_h, a_l, b_h,\) and \(b_l\), where \(a_h\) is the high-order bit of \(a\) and \(a_l\) is the low-order bit, and similarly for \(b\). Label your outputs \(o_0, o_1, o_2,\) and \(o_3\), where \(o_i\) is the output corresponding to the bit valued as \(2^i\).

You can use boxes to represent half-adders, instead of drawing out their logic gates. Label the sum and carry outputs as \(S\) and \(C\). Or, if you prefer, you can just use the XOR and AND gates that compose them.

Recall that binary multiplication works similarly to base 10 – you’ll need to multiply by the one’s digit, multiply by the two’s digit, and add.
Note that you can multiply two binary digits using just one logic gate (and you should, for simplicity).

Solution: