CSE 140 Homework Two

May 5, 2016

Only Problem Set Part B will be graded. Turn in only Problem Set Part B which will be due on May 18, 2016 (Wednesday) at 1:00pm.

1 Problem Set Part A

All questions in this part are from Roth&Kinney, 7th Edition.

- 4.40, 4.44
- 5.5, 5.8, 5.10, 5.11, 5.12, 5.13, 5.20, 5.28, 5.30, 5.32, 5.33, 5.34, 5.42
- 6.6, 6.9, 6.11, 6.16, 6.17, 6.24
- 7.1, 7.4, 7.5, 7.6, 7.8, 7.9, 7.10, 7.11, 7.14, 7.26, 7.27, 7.28, 7.41, 7.42
- 8.2, 8.6, 8.7, 8.8
- 13.3(a)(b), 13.4(a), 13.8(a), 13.11(a)
- 14.4, 14.12, 14.13, 14.17, 14.23
2 Problem Set Part B

1. Karnaugh Maps

In an unfortunate turn of events, one of your coworkers manages to destroy almost all of the work involved in your company’s latest CPU design. The only readable papers left are a few Karnaugh maps, but even those have been damaged. In his desperation, your coworker turns to you for help.

The problem is that this coworker isn’t very good at Karnaugh maps: on several occasions you’ve seen his row and column headers violating the adjacency constraint for K-maps. On the other hand, someone reputable did the logic minimization and wrote down notes about the minimized function, so at least you have some useful info.

For this question, inverters do not count towards gate count or input count.

(Part A) You decide to go with the least damaged Karnaugh map first. The column headers have been rendered unreadable, but at least you know that this map was implemented as a sum-of-products, using 4 AND/OR gates with 12 inputs collectively.

\[
\begin{array}{c|cccc}
ab\backslash cd & ? & ? & ? & ? \\
00 & & & & \\
01 & 1 & 1 & 1 & \\
11 & 1 & 1 & 1 & \\
10 & & & & \\
\end{array}
\]

Do the missing headers obey the adjacency constraints for Karnaugh maps? If not, give a set of headers that fits the given constraints.

(Part B) Moving on, the next map you look at has had its row headers torn off. Some notes next to the Karnaugh map indicate that the function for this map uses 2 AND/OR gates, with 4 inputs collectively, in its minimal form, but you can’t tell if this map was implemented as a sum-of-products or as a product-of-sums.

\[
\begin{array}{c|cccc}
ab\backslash cd & 00 & 01 & 11 & 10 \\
? & 0 & 0 & 0 & 0 \\
? & 1 & 1 & 1 & 1 \\
? & 0 & 0 & 0 & 0 \\
? & 1 & 0 & 1 & 1 \\
\end{array}
\]

Do the missing headers obey the adjacency constraints for Karnaugh maps? If not, give one set of headers that fits the given constraints.
(Part C) The last map makes you want to strangle your co-worker, because it shouldn’t be possible to destroy a Karnaugh map so thoroughly. Not only are some of the headers missing, but some of the entries have been destroyed as well! Luckily, you remember that this function was implemented as a **product-of-sums**, using 4 AND/OR gates and 9 inputs.

\[
\begin{array}{cccc|c}
ab & \? & \? & \? & \? \\
00 & 0 & 0 & 0 & 1 \\
01 & 0 & ? & 0 & ? \\
11 & ? & 0 & 0 & 1 \\
10 & 1 & 1 & 1 & ? \\
\end{array}
\]

Do the missing headers obey the adjacency constraints for Karnaugh maps? If not, give one set of headers that fits the given constraints.

Are the missing entries recoverable? If so, write in the entries in the K-map below.

\[
\begin{array}{cccc|c}
ab & \? & \? & \? & \? \\
00 & 0 & 0 & 0 & 1 \\
01 & 0 & ? & 0 & ? \\
11 & ? & 0 & 0 & 1 \\
10 & 1 & 1 & 1 & ? \\
\end{array}
\]
2. (Petrick’s Method)

As you recall from lecture, Petrick’s Method is an algorithm that is able to identify the minimal cover of a given prime implicant table. It does so by first casting the implicant cover as a product-of-sums problem and then converting this formulation into a sum-of-products via distributivity and absorption.

(Part A) As you recall, the main benefit of Petrick’s Method over the Quine-McCluskey heuristics is that it is able to resolve the minimal solution of a cyclic core. As a reminder, an example of a cyclic core is shown below:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_4$</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_5$</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_6$</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

The Prime Implicant table shown above converts to the following product-of-sums Petrick’s Method formulation:

$$(P_5 + P_6)(P_3 + P_4)(P_1 + P_2)(P_2 + P_3)(P_4 + P_5)(P_1 + P_6)$$

Please use Petrick’s Method to identify all of the prime implicant sets that can form a minimal cover.
(Part B) While Petrick’s Method is usually performed on a cyclic core, it was mentioned in class that Petrick’s Method can be performed at any point after the generation of prime implicants, even if a cyclic core has not yet been reached. This implies a tightly coupled relationship between Petrick’s Method and the Quine McCluskey row/column domination heuristics. As a reminder, the Quine McCluskey steps are reproduced below:

1. If an implica(nt/ate) is essential, it must be in our final cover, so the implica(nt/ate) can be removed, and its (min/max)terms can be crossed out.

2. If a row of prime implica(nts/ates) is a subset of another row, the subset row does not have to participate in our final cover, so it can be crossed out.

3. If a column of (min/max)terms is a superset of another column, the superset column is redundant and can be crossed out.

Keeping this in mind, we ask that you look at the following sets of Petrick’s Method formulations and simplifications. Please identify the numeric values for the variables $a, b, c,$ and $d$ that allow for the simplifications to be equivalent. Alternatively, if no values for $a, b, c,$ and $d$ allow for the simplification to be valid, please say so. Please provide your reasoning for either case.

i. $P_aP_b(P_2 + P_4)(P_1 + P_3)(P_3 + P_5)(P_1 + P_2) = P_1P_2(P_c + P_d)$

ii. $(P_a + P_2)(P_b + P_4 + P_5)(P_1 + P_3)(P_1 + P_c) = (P_1 + P_2)(P_1 + P_d)(P_1 + P_4)$

iii. $(P_1 + P_2)(P_2 + P_3)(P_3 + P_4)(P_4 + P_5)(P_5 + P_6)(P_1 + P_6) = P_aP_b(P_c + P_5)(P_5 + P_d)$

iv. $(P_a + P_2 + P_3)(P_1 + P_b + P_4)(P_3 + P_4 + P_5)(P_1 + P_5) = (P_1 + P_c)(P_d + P_4)(P_3 + P_4 + P_5)(P_1 + P_5)$
(Part C) One of your classmates has made the observation that splitting up the cyclic core in the manner shown below allows you to generate the same minimal solution(s) that you obtained in (Part A):

Please apply the row/column dominance heuristics on the two Prime Implicant tables, and let us know which solution(s) each table generates. How do these correspond to the solutions you found in (Part A)?

(Part D) As we reminded you in (Part B), all applications of Quine-McCluskey have a theoretical basis in Petrick’s Method, meaning that if your classmate’s observation in (Part C) is correct, there must be a direct translation into Petrick’s Method. **Explain the boolean transformation on the expression in (Part A) that corresponds to splitting up the cyclic core in the Prime Implicant table as shown in (Part C).**
3. Deciphering Characteristic Equations

As you have by now done many times in your homework, you can easily go from a function in the form of a list of minterms or in a K-Map to a sum-of-products or product-of-sums. Likewise, you can create a list of minterms or a K-Map from the product-of-sums or sum-of-products. However, any don't cares in the original function will either be assigned a fixed value of 1 or 0 when coming up with the product-of-sums or sum-of-products. As a result, much of the information about what input combinations to the original function result in a don't care gets lost. However, if all of the terms are minimal implicants or minimal implicates, it is possible to figure out which outputs we are certain about, and which ones could be a don't care.

To illustrate how this can be done, let’s take the simple example sum-of-products $X + Y$ containing minimal implicants. If we assume that there are no don’t cares in the original function, we would end up with a K-Map that looks like the following:

\[
\begin{array}{c|cc}
X \backslash Y & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

As can be seen from the K-Map, the minterm for the input combination $(X = 1, Y = 1)$ is covered by both implicants. However, if the value for the input combination $(X = 1, Y = 1)$ was originally a don’t care, in the minimal cover we would end up with the exact same set of implicants. That means that for the input combination $(X = 1, Y = 1)$, the original K-Map could have contained either a 1 or a don’t care. Now, let’s see what happens if the input combination $(X = 1, Y = 0)$ is instead replaced with a don’t care. With this minor alteration, the K-Map would become:

\[
\begin{array}{c|cc}
X \backslash Y & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & \text{dc} & 1 \\
\end{array}
\]

Since we are going for minimal implicants, we would get the implicant $Y$ to cover the right-most column, and then we are done, since our newly introduced don’t care does not need to be covered. The resulting sum-of-products is therefore just $Y$, which is different from the sum-of-products we were originally given; since the original sum-of-products was reportedly in minimal form, we know that an input combination of $(X = 1, Y = 0)$ can’t be a don’t care, as that would have led to the $X + Y$ sum-of-products not to be the minimal implicant cover. By similar reasoning, an input of $(X = 0, Y = 1)$ can’t be a don’t care either.

Now, what happens if the input $(X = 0, Y = 0)$ were a don’t care? In this case, the resulting K-Map would look like:

\[
\begin{array}{c|cc}
X \backslash Y & 0 & 1 \\
\hline
0 & \text{dc} & 1 \\
1 & 1 & \text{1 or dc} \\
\end{array}
\]

The minimal implicant in this case would thus contain all of the entries in the function (regardless of whether $(X = 1, Y = 1)$ is a 1 or a don’t care), so for all input combinations the output will be a 1, resulting in a minimal sum-of-products representation of 1 instead of $X + Y$. It should thus be evident that the original K-Map that gave rise to the minimal implementation of $X + Y$ could be either of the two K-Maps shown summarily below:

\[
\begin{array}{c|cc}
X \backslash Y & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & \text{1 or dc} \\
\end{array}
\]

7
Now, after all this talk about the sum-of-products with minimal implicants you may be wondering what happens if the product-of-sums with the minimal implicates formulation of a function is given instead. Well, if we replace all of the 1’s with 0’s and vice versa in the previous examples, you will notice that the same lines of reasoning can be applied. For the minimal product-of-sums derivation of \((X')(Y')\), the possible designations (i.e. 0, 1, and/or dc) for each entry in the original function are shown in the K-Map below. As can be seen, the entry for \((X = 1, Y = 1)\) could be either a 0 or a don’t care.

\[
\begin{array}{c|cc}
 X \backslash Y & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \text{ or dc}
\end{array}
\]

(Part A) With your newfound understanding of how don’t cares can disappear when writing a function in its product-of-sums or sum-of-products formulation, please indicate which of the K-Maps shown below could be the original function that results in the minimal sum-of-products form of \(Y' + Z\) by circling all of the ones that are possible.

\[
\begin{array}{c|cccc}
 X \backslash YZ & 00 & 01 & 11 & 10 \\
0 & dc & 1 & 1 & 0 \\
1 & dc & 1 & dc & 0
\end{array}
\]

\[
\begin{array}{c|cccc}
 X \backslash YZ & 00 & 01 & 11 & 10 \\
0 & 1 & dc & dc & 0 \\
1 & dc & dc & 1 & 0
\end{array}
\]

\[
\begin{array}{c|cccc}
 X \backslash YZ & 00 & 01 & 11 & 10 \\
0 & 1 & dc & 1 & 0 \\
1 & 1 & 1 & dc & 0
\end{array}
\]

\[
\begin{array}{c|cccc}
 X \backslash YZ & 00 & 01 & 11 & 10 \\
0 & 1 & dc & 1 & dc \\
1 & dc & dc & dc & 0
\end{array}
\]

(Part B) Now that you have figured out how to identify which observed outputs could potentially be don’t cares, let’s consider what additional information can be gleaned when a product-of-sums and a sum-of-products for the same function are given, both of which contain only their minimal implicants and implicates. Whereas with only the sum-of-products or product-of-sums form of a function given you are restricted to idle speculation on whether or not a particular output is a don’t care or a fixed value, when both forms are provided you can tell with certainty that certain outputs are definitely a don’t care when the two forms of the same original function have different outputs. For the minimal sum-of-products and minimal product-of-sums forms of the same function given below, please fill in each entry of the K-Map with all possible values that entry could have. (Remember that for some entries that appear to have a value of 1 or 0, a don’t care could also be a possible value in the original function.)

\[
\begin{array}{c|cccc}
 X \backslash YZ & 00 & 01 & 11 & 10 \\
0 & \\
1 & 
\end{array}
\]

SOP: \(Y' + XZ\)

POS: \((X + Z')(Y' + Z)\)
(Part C) After hearing of your expertise with reconstructing the original functions from sets of equations containing minimal implicants and implicates, two of your classmates come to you seeking assistance with their tale of woe. They had been designing a new form of flip-flop; however, after a late night studying for an exam they woke up only to discover that their cat was eating the notes detailing how the flip-flop state changes for different combinations of inputs. After salvaging the remainder of the notes from the cat, they discover to their despair that the only partially readable portion of their notes is the characteristic equation for their flip-flop design, in both minimal sum-of-products ($C_1$) and minimal product-of-sums ($C_2$) form.

\[
C_1 : Q_{next} = I'_0 + Q_{curr}I_1 \\
C_2 : Q_{next} = (I_1)(Q_{curr} + I'_0)
\]

To help identify for what input combinations the output of the characteristic equations will differ, your classmates ask you to draw out a separate K-Map for $C_1$ and $C_2$.

\[
\begin{array}{c|cc|cc}
Q_{curr} \setminus I_0 & 00 & 01 & 11 & 10 \\
\hline
0 & & & & \\
1 & & & & \\
\end{array}
\]

\[
\begin{array}{c|cc|cc}
Q_{curr} \setminus I_0 & 00 & 01 & 11 & 10 \\
\hline
0 & & & & \\
1 & & & & \\
\end{array}
\]

(Part D) After seeing the discrepancies between the two forms of the original function, your classmates ask about which of the entries could have been don’t cares. To help them out, they ask you to fill in a K-Map that shows in each entry the possible values that entry could contain. (Remember that for some entries that appear to have a value of 1 or 0, a don’t care could also be a possible value in the original function.)

\[
\begin{array}{c|cc|cc}
Q_{curr} \setminus I_0 & 00 & 01 & 11 & 10 \\
\hline
0 & & & & \\
1 & & & & \\
\end{array}
\]

(Part E) After recovering from the trauma of studying for a midterm, your classmates recall that their flip-flop design only had 2 don’t care conditions. From this additional information you quickly realize which of the don’t cares you’ve identified are part of their flip-flop design, and write the updated K-Map with the correct values for your classmates.

\[
\begin{array}{c|cc|cc}
Q_{curr} \setminus I_0 & 00 & 01 & 11 & 10 \\
\hline
0 & & & & \\
1 & & & & \\
\end{array}
\]

(Part F) Unfortunately, your classmates are still unable to recall how their flip-flop was supposed to operate, and ask you to fill out the characteristic table for their flip-flop, and to give descriptive names to the two inputs, $I_0$ and $I_1$. (Note that entries for $Q_{next}$ are not limited to the values of 0 and 1. They may be written in terms of a variable such as $Q_{curr}$; certain entries may also be invalid if the behavior involves only don’t cares.)

\[
\begin{array}{c|cc|c}
I_0 & I_1 & Q_{next} \\
\hline
0 & 0 & \\
0 & 1 & \\
1 & 0 & \\
1 & 1 & \\
\end{array}
\]