HW SET-1
No 1

• (a) No. Edge coverage subsumes node coverage. Therefore, any test suite (of any program) that gives full edge coverage will also give full node coverage.

• (b) Let me first note that node coverage does not subsume edge coverage. This implies that there exists at least one program P and one test set T for P that provides full node coverage on P, but T does not give full edge coverage on P. However, this certainly does not hold for all programs, such as the above program. For the above program: any test set that gives full node coverage (on the above program) will also give full edge coverage. Note also that a test-path is required to end in the terminal node. So the answer is “No”.

• (c) Yes. Consider the following test set, consisting of a single test case:

  \{ [1,2,3,2,4] \} \rightarrow \text{it misses the pairs } [1,2]-[2,4] \text{ and } [3,2]-[2,3]

• (d) 2 e.g. the test set: \{ [1,2,4] , [1,2,3,2,3,2,4] \}
No 2

• (a) TR1 requires us to cover all nodes in G, and TR2 all edges. As discussed in No-1, in general they are not equivalent (they do not subsume each other). The above program demonstrates this. E.g. the test set containing a single test path [1,2,3,2,4,5,6,1,7] covers all nodes, but it misses the edge [4,6].

• (d) there are 15 prime paths (pps).
  [1,2,4,6,1]  [1,2,4,5,6,1]
  [2,3,2]  [2,4,6,1,2]  [2,4,5,6,1,2]
  [3,2,3]  [3,2,4,6,1,7]  [3,2,4,5,6,1,7]  \(\rightarrow\) note that reds begin in 3, and end in terminal 7
  [4,6,1,2,4]  [4,5,6,1,2,4]  [4,6,1,2,3]  [4,5,6,1,2,3]  \(\rightarrow\) note that reds begin in 4, and end in non-terminal 3!
  [5,6,1,2,4,5]
  [6,1,2,4,6]  [6,1,2,4,5,6]

• (b) There are 2 pps that start in 1
• (c) There are 6 pps that pass 3
No 3 & 4

Notation alternatives:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>returnk</td>
<td>retval</td>
</tr>
<tr>
<td>p &amp;&amp; q</td>
<td>p \land q</td>
</tr>
<tr>
<td>p</td>
<td></td>
</tr>
<tr>
<td>x == y</td>
<td>x = y</td>
</tr>
<tr>
<td>Exists(a, k \rightarrow a[k]=0)</td>
<td>(\exists k: 0\leq k&lt;#a : a[k]=0)</td>
</tr>
<tr>
<td>Forall(a, k \rightarrow a[k]=0)</td>
<td>(\forall k: 0\leq k&lt;#a : a[k]=0)</td>
</tr>
</tbody>
</table>

• No 3:

\{ * true * \}
int MIN(int x, int y)
\{ * (retval==x || retval==y) && retval \leq x && retval \leq y * \}

• No 4:

\{ * true * \}
boolean isMIN(int m, int x, int y)
\{ * retval == ((m==x || m==y) && m\leq x && m\leq y) * \}
• [0,1,0,2] **tours** [0,2] (so, it also tours [0,2] with side trip and detour)
• [0,1,2] does not tour [0,2], not even with side trip.
• [0,1,2] tours [0,2] **with detour**
• [0,1,2,0,1,0,2] does not tour [2,0,2]. However, it does tour [2,0,2] **with side trip** (so, also with detour).
HW SET-2
(a) The test set \{ (0,1,2), (-1,-1,-1) \} does not give full ECC. For example, it misses blocks b1 and b2 of Side-1, and blocks b1 and b3 of Side-2.

(b) The test set \{ (0,1,2), (2,0,1), (1,2,0), (-1,-1,-1) \} does give full ECC.
(c) The following 16 test cases give full PWC:

- \((2,2,2)\) \((2,1,-1)\) \((2,0,0)\) \((2,-1,1)\)
- \((1,2,1)\) \((1,1,2)\) \((1,0,-1)\) \((1,-1,0)\)
- \((0,2,0)\) \((0,1,1)\) \((0,0,2)\) \((0,-1,-1)\)
- \((-1,2,-1)\) \((-1,1,0)\) \((-1,0,1)\) \((-1,-1,2)\)

Note: the solution is not unique btw... To keep that in mind 😊

(d) I manage it with 16, let me know if you can come up with less.
Consider again the above partitioning of TriTyp. This time we want to apply MBCC, using the following choice of base blocks: \{b_1,b_2\} for q_1, \{b_1\} for q_2, and \{b_1\} for q_3 (for your convenience: b_1 represents greater than 1, and b_2 represents exactly 1).

- Consider the following test set T. It gives full MBCC. Furthermore, it is minimal (the smallest test set that gives MBCC).
  - Base tests: (2,2,2) and (1,2,2)
  - Additionals:
    - wrt to q_1: (0,2,2) and (0,2,2) \rightarrow \text{dup}
      - (-1,2,2) and (-1,2,2) \rightarrow \text{dup}
    - wrt to q_2: (2,1,2) and (1,1,2)
      - (2,0,2) and (1,0,2)
      - (2,-1,2) and (1,-1,2)
    - wrt to q_3: (2,2,1) and (1,2,1)
      - (2,2,0) and (1,2,0)
      - (2,2,-1) and (1,2,-1)
  - Because of two duplicates, effectively you only have 16 test cases.

- (a) Yes, MBCC subsumes ECC.
- (b) No.
- (c) MBCC does not require you to cover combinations among non-base blocks. Since T is minimal, it won’t cover such non-base blocks combinations, including pairs over them. For example, the pairs (b_3,b_3,x) or (b_3,x,b_3) will be missed by T due to its minimality towards MBCC. So ... the answer is “No”. Note that a test set T’ that gives full MBBC but is not minimal might give full PWC. A trivial example is if T’ simply includes all combinations over all blocks. MBBC may not give full PWC. The above test set is an example of the latter case (it does not give you full PWC).
- (d) According to the formula given in A&O, we should get at most 2(\text{base}) + 4 + 6 + 6 = 18. It could be less as some test cases may turn out to be duplicates. There is no formula known to calculate the actual minimum size, other than to actually calculating it by quantifying over all possibilities. For the above given test set T, it has 18 test cases. However, two are duplicates. So the size of that T is 16.
No 3 & 4

- No 3, using in-code notation:

\[
\begin{align*}
\{ & \text{* } x>0 \text{ } \&\& \text{ } y>0 \text{ } \&\& \text{ } z>0 \text{ } \} \\
\text{boolean TRIANGLE}(\text{int } x, \text{ int } y, \text{ int } z) \\
\{ & \text{* retval } == (x+y>z \text{ } \&\& \text{ } x+z>y \text{ } \&\& \text{ } y+z>x) \text{ } \}
\end{align*}
\]

- No 4, using in-code notation:

\[
\begin{align*}
\{ & \text{* a\neq null } \text{ } \&\& \text{ } a.\text{size}>0 \text{ } \} \\
\text{int getMIN(\text{int[ ] a})} \\
\{ & \text{* exists( } a, \text{ } i \rightarrow \text{ retval}==a[i] ) \text{ } \&\& \text{ } \forall i (a, \text{ } i \rightarrow \text{ retval}\leq a[i] ) \text{ } \}
\end{align*}
\]

Using predicate logic notation:

\[
\begin{align*}
\{ & \text{* a\neq null } \text{ } \&\& \text{ } a.\text{size}>0 \text{ } \} \\
\text{int getMIN(\text{int[ ] a})} \\
\{ & \text{ ( } \exists i : 0\leq i < \text{a.length} : \text{ retval}==a[i] ) \text{ } \&\& \text{ } ( \forall i : 0\leq i < \text{a.length} : \text{ retval}\leq a[i] ) \text{ } \}
\end{align*}
\]
HW SET-3
(a) Is it possible that a program contains a prime path which is not a du-path of any variable? Yes. Example: `getx() { return x }`, which does not define any variable, and hence has no du-path.

(b) Suppose a program P contains a variable x. Is it possible that it contains a du-path for x which is not a simple path? No. Any du-path is by definition a simple path.

(c) Suppose [n1,n2,n3] is a du-path for x. Where do writes to x take place? In any case in the node n1. By definition, n3 will contain a use of x. But... n3 may also contain a def of x, but if this is the case we assume this def to occur, in the code fragment represented by n3, after the use of x in n3.
(a) Let us first list down all the du-paths of x. See below. I organize them in def-pair sets, and I will also give names to the paths so that I can refer to them later:

- du(0,7,x) = {dp1} where dp1 is the path [0, 1, 7]
- du(0,5,x) = { dp2} where dp2 is the path [0, 1, 2, 4, 5]
- du(3,5,x) = {dp3} where dp3 is [3, 2, 4, 5]
- du(3,7,x) = {dp4,dp5} where dp4 is [3, 2, 4, 6, 1, 7], and dp5 is [3, 2, 4, 5, 6, 1, 7]

So... in total there are 5 du-paths of x.

(b) Roughly said, full all-defs coverage over x requires you to cover all “def”-locations of x. More precisely, you are require to cover all (non-empty) def-path sets of x. These are all x’s def-path sets:

- du(0,x) = {dp1,dp2}
- du(3,x) = {dp3,dp4,dp5}

Notice that du(0,x) = du(0,5,x) U du(0,7,x), and du(3,x) = du(3,5,x) U du(3,7,x). The TR for “full all-defs coverage over x” consists of one chosen path from each of the above def-path sets. For example, this TR could be {dp1,dp3}. But TR = {dp2,dp5} is also good. Regardless the choice, we have two def-path sets, so the TR will always contain two paths. (so the final answer is “2”).

(c) Roughly, full all-uses coverage over x requires you to cover all pairs of def and use locations of x. More precisely, you need to cover all (non-empty) def-pair sets as listed down in (a). The TR will consists of one chosen path from each def-pair. For example it could be TR={dp1,dp2,dp3,dp4}. But it could also be TR={dp1,dp2,dp3,dp5}. In this case, this is the only two possible TRs. The size of this TR is 4 (because there are 4x def-pairs of x). So the final answer is also “4”.
Now we can easily answer each question:

(a) There are 7 du-paths for x, 2 for y, and 2 for a.
(b) To get all du-path coverage for y you need to cover all du-paths for y. There are 2.
(c) To get full all-uses coverage for a you need to cover all du-pairs of a. You end up with two paths to cover. See also the explanation of no 2.
(d) 7. See also the explanation of no 2.

Let us first list down all the du-paths of the variables involved. Note that a du-path must also be a simple path. See below

**du-paths of x, organized in du-pairs:**
- \(du(0,1,x) = \{01\}\)
- \(du(0,7,x) = \{017\}\)
- \(du(0,4,x) = \{0124\}\)
- \(du(3,7,x) \rightarrow \text{none}\)
- \(du(3,4,x) = \{324\}\)
- \(du(6,1,x) = \{61\}\)
- \(du(6,7,x) = \{617\}\)
- \(du(6,4,x) = \{6124\}\)

**du-paths of y, organized in du-pairs:**
- \(du(0,2,y) = \{012\}\)
- \(du(3,2,y) = \{32\}\)

**du-paths of a, organized in du-pairs:**
- \(du(4,6,a) = \{46\}\)
- \(du(5,6,a) = \{56\}\)
In in-code notation:

```java
boolean COMMON(int[] a, int[] b)
{*
  retval == exists(a, i→ exists(b, k→ a[i]==b[k])) *
}
```

In predicate logic notation:

```java
boolean COMMON(int[] a, int[] b)
{*
  retval = (∃i: 0≤i<a.length : (∃k: 0≤k<a.length : a[i]=b[k])) *
}
```
No 5

- In in-code notation:
  
  ```
  {a \neq \text{null} \land \forall (a, i \rightarrow \forall (a, k \rightarrow \text{imp}(i \neq k, a[i] \neq a[k])))}
  
  \text{int[ ] SORT(int[ ] a)}
  
  {\forall (\text{retval}, i \rightarrow \exists (a, k \rightarrow \text{retval}[i]==a[k]))}
  
  \text{&& \forall (a, i \rightarrow \exists (\text{retval}, k \rightarrow a[i]==\text{retval}[k]))}
  
  \text{&& \forall (\text{retval}, i \rightarrow \forall (\text{retval}, k \rightarrow \text{imp}(i<k, \text{retval}[i] < \text{retval}[k])))}
  
  ```

- In predicate logic notation:

  ```
  {a \neq \text{null} \land (\forall i: 0 \leq i < a.\text{length}: (\forall k: 0 \leq k < a.\text{length}: i \neq k \Rightarrow a[i] \neq a[k])} \land
  
  \text{int[ ] SORT(int[ ] a)}
  
  {\forall i: 0 \leq i < \text{retval}.\text{length}: (\exists k: 0 \leq k < a.\text{length}: \text{retval}[i]==a[k])}
  
  \land (\forall i: 0 \leq i < a.\text{length}: (\exists k: 0 \leq k < \text{retval}.\text{length}: a[i]==\text{retval}[k])}
  
  \land (\forall i: 0 \leq i < \text{retval}.\text{length}: (\forall k: 0 \leq k < \text{retval}.\text{length}: i < k \Rightarrow \text{retval}[i] < \text{retval}[k]))
  ```
No 6
The data flows of `stut()` and `checkDups()`

The picture below shows again the control flow graphs of the two programs, `sust` and `checkDups`, decorated with the def and use information of variables involved. There are two call sites of interest, marked blue in the picture. The **coupling vars** are: `linecnt/line, lastdelimit, curWord`. We will first list all the coupling paths for each of these variables, for each call site.

**coupling vars:**
- `linecnt/line`
- `lastdelimit`
- `curWord`

**Call Sites:**
1. `call checkDups(linecnt)`
2. `call checkDups(linecnt)`
3. `call checkDups(linecnt)`

**Variables:**
- `linecnt`
- `line`
- `lastdelimit`
- `curWord`
- `prevWord`

**Variables Usage:**
- `def`: Defines a variable.
- `use`: Uses a variable.

**Paths:**
1. `def = {inFile}`
2. `use = {i, inLine}`
3. `use = {inLine, i}`
4. `use = {c}`
5. `use = {linecnt}`
6. `use = {linecnt}`
7. `use = {curWord, c}`
8. `use = {linecnt}`
9. `exit`
The **coupling vars** are `linecnt/line`, `lastdelimit`, `curWord`. Nodes where we have either a last def or first use of these variables, in either side of the call, have been marked green above, just to help you locating them. I will write s0, s1 etc to mean the nodes of “stut”, and c0, c1 etc to mean the nodes of checkDups.

The coupling paths of these variables for the call site at location s8 are listed below. Do note that a couple path is a du-path, and therefore must also be a simple path.

- Since `linecnt` is just pass through as the parameter `line` to checkDups, for deciding first-uses at checkDups I treat them as the same. However, this parameter is passed by value, so the parameter `line` is not a coupling-var towards `stut`.
- `linecnt` has only one coupling path: `[s6,s8,c0,c1,c3,c4,c5]`
- `lastdelimit` also has one coupling path: `[s7,s2,s6,s8,c0,c1]`
- `curword` has two coupling paths: `[s7,s2,s6,s8,c0,c1,c3,c4]`, `[c7,c8,s8,s1,s2,s3,s4,s7]`
The coupling vars are \textit{linecnt/line, lastdelimit, curWord}. The coupling paths of these variables for the call site at location s5 are listed below:

- \textit{linecnt} has in any case this coupling path: \{s0,s1,s2,s3,s4,s5,c0,c1,c3,c4,c5\}. Additionally, there is a def of linecnt at s6. The information of this def flows however through the call site s8. However, since we know the call does not affect linecnt, we will also count \{s6,s8,s1,s2,s3,s4,s5, c0,c1,c3,c4,c5\} as a coupling path for linecnt.
- \textit{lastdelimit} also has one coupling path: \{s7,s2,s3,s4,s5, c0,c1,c3,c4\}
- \textit{curWord} has two coupling paths: \{s7,s2,s3,s4,s5,c0,c0,c1,c3,c4\} , \{c7,c8,s5,s2,s3,s4,s7\}
Now we can answer the questions:

(a) For call site s8, we have:
   - linecnt: 1 coupling path
   - lastdelimit: 1 coupling path
   - curword: 2 coupling paths

(b) For call site s5, we have:
   - linecnt: 2 coupling paths
   - lastdelimit: 1 coupling path
   - curword: 2 coupling paths

(c) Well, for each call site, we only have one coupling path for lastdelimit. So, to get full "All Coupling Uses Coverage" (ACUC) over lastdelimit, these are the only paths need to be covered. So, the final answer is “1 per call site”.

(d) For each call site, curword has two coupling paths. Notice that each has different last-def locations. So, for full "All Coupling Uses Coverage" we will have to cover them all. So, the final answer is “2 per call site”. Btw, notice that even for the weaker “All CouplingDefs Coverage” we have to cover both coupling paths of curword (per call site).
Older HWs
3.1, no 1

- Clauses of \(( f \leq g) \land (X > 0)) \lor (M \land (e < d + c))\):
  - \(f \leq g\)
  - \(X > 0\)
  - \(M\)
  - \(e < d + c\)
3.2 no 1, pred 1: $p = a \land (\neg b \lor c)$

(b) activations
- activation of $a$: if $\neg b \lor c$
- activation of $b$: if $a \land \neg c$
- activation of $c$: if $a \land b$

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>activated</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a,c</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>b,c</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>a,b</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>0</td>
</tr>
</tbody>
</table>

(c) truth table
3.2 no 1, pred 1: \( p = a \land (\neg b \lor c) \)

(d) Identify all pairs of rows from your table that satisfy general active clause coverage (GACC) with respect to each clause.

(11 pairs)

For b and c there is only one option (see the table)

For a there are 9 pairs: \{1,3,4\} x \{5,7,8\}

(e) Identify all pairs of rows from your table that satisfy correlated active clause coverage (CACC) with respect to each clause.

The same pairs as GACC.

(f) Identify all pairs of rows from your table that satisfy restricted active clause coverage (RACC) with respect to each clause.

Again for b and c, we only have one option.

For a we have three pairs: (1,5) (3,7) and (4,8)
3.2 no 1, 
\[ p = a \land (\neg b \lor c) \]

(g) Identify all 4-tuples of rows from your table that satisfy general inactive clause coverage (GICC) with respect to each clause. Identify any infeasible GICC test requirements.

For a, it is not possible to make \( p=1 \).

For b, (<1,3>, and any combination of <x,y> from \{5,6\} \times \{7,8\} ) \rightarrow 4 possibilities.

For c (<3,4> and and any combination of <x,y> from \{5,7\} \times \{6,8\} ) \rightarrow 4 possibilities.

(h) Identify all 4-tuples of rows from your table that satisfy restricted inactive clause coverage (RICC) with respect to each clause. Identify any infeasible RICC test requirements.

For a it is of course also not possible. For b: (<1,3>,<5,7>) and (<1,3>,<5,7>) For c: (<3,4>,<5,6>) and (<3,4>,<7,8>)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>activated</th>
<th>inact</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a,c</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>b,c</td>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>a,b</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>b,c</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>a,b,c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that on combinations where a clause is not active, it follows that it is then inactive.
3.2 no 1, pred 1: \( p = a \rightarrow (b \rightarrow c) \)

(b) activations
- activation of a: if \( \sim(b \rightarrow c) = b \land \sim c \)
- activation of b: if \( a \land \sim c \)
- activation of c: if \( a \land b \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>activated</th>
<th>inact</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>c</td>
<td>a,b</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>a,b,c</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>b</td>
<td>a,c</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

(c) truth table
### 3.2 no 1, pred 1: \( p = a \rightarrow (b \rightarrow c) \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>activated</th>
<th>inact</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>c</td>
<td>a,b</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>a,b,c</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>b</td>
<td>a,c</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**GACC pairs.**
For each there is only one option.

**CACC and RACC pairs are the same as GACC.**

**GICC 4-tuples.**
None is possible, because we cannot make \( p=-0 \).

Then none is also possible for RICC.
3.2 no 1, pred 1: \[ p = a \leftrightarrow (b \land c) \]

(b) activations
- activation of a: always
- activation of b: if c
- activation of c: if b

(c) truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>activated</th>
<th>inact</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>a,c</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>a,b</td>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>a,b,c</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>a,c</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a,b</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>b,c</td>
<td>1</td>
</tr>
</tbody>
</table>
3.2 no 1, pred 1: \[ p = a \leftrightarrow (b \land c) \]

### GACC pairs. (24 pairs)
- For a: \{1,2,3,4\} \times \{5,6,7,8\} (16 pairs)
- For b: \{1,5\} \times \{3,7\}
- For c: \{1,5\} \times \{2,6\}

### CACC pairs (14 pairs)
- For a: \((1,5) +\{2,3,4\} \times \{6,7,8\}\)
- For b: \((1,3), (5,7)\)
- For c: \((1,2), (5,6)\)

### RACC pairs (8 pairs)
- For a: \((1,5),(2,6),(3,7),(4,8)\)
- For b: \((1,3), (5,7)\)
- For c: \((1,2), (5,6)\)

### GICC tuples
- For a is not possible (cannot be inactivated)
- For b: \((<2,4>,<6,8>)\)
- For c: \((<3,4>,<7,8>)\)

### RIC tuples: same as above.
Section 3.6, No 1

answer the same questions as 4a,b,c,d,e

- \( f = abc + abc \)

(a) Karnaugh map of \( f \) is shown. That of \( \neg f \) is where we reverse 0/1 in \( f \)'s values.

(b) \( f \) is already minimal; but this one too:
\[
\begin{align*}
f &= bc \\
\end{align*}
\]

For \( \neg f \) for example: \( bc + b \)

(c) A test-test that satisfies implicant coverage, wrt to the above choices of DNFs: 010, 011, 000

(d) For UTPC: same as (c) will do

(e) CUTPNFP, only one implicant in \( f \), namely bc; for each clauses here:
   - for clause b: 010 (UTP) and 000 (NFP)
   - for clause c: 010 (UTP) and 011 (NFP)
Section 3.6. No 4 (a,b,c,d,e)

- \(f = a c d + cd + bcd\)

- (a) Karnaugh map of \(f\) is shown. That of \(\sim f\) is where we reverse 0/1 in \(f\)'s values.

- (b) \(f\) is already minimal; but this one too: \(f = a c + cd + bd\)

For \(\sim f\): \(ad + cd + bcd\)

- (c) For implicant cov, e.g: 
  \(1101, 0010, 0011 \rightarrow f\)
  \(1010, 1011 \rightarrow \sim f\)

- (d) CUTP: orange + red boxes in the table above. Those are the UTPs of the corresponding implicants in \(f\) and \(\sim f\).
Section 3.6.No 4 (a,b,c,d,e)

• \( f = \overline{a} \overline{c} d + \overline{c} d + b c d \)

• Using this equivalent rep of \( f \), which is also minimal

\[
\begin{align*}
    f &= \overline{a} \overline{c} + \overline{c} d + b d \\
\end{align*}
\]

• CUTPNFP

For each implicant, we need to cover each clause in the implicant:

Implicant \( a \overline{c} \); we have two clauses:

\[
\begin{align*}
    a & : 0000 \text{ (UTP)} + 1000 \text{ (NFP)} \\
    c & : 0000 \text{ (UTP)} + 0010 \text{ (NFP)}
\end{align*}
\]

Implicant \( \overline{c} d \); we have two clauses:

\[
\begin{align*}
    c & : 1001 \text{ (UTP)} + 1011 \text{ (NFP)} \\
    d & : 1001 \text{ (UTP)} + 1000 \text{ (NFP)}
\end{align*}
\]

Implicant \( b d \); we have two clauses:

\[
\begin{align*}
    b & : 0111 \text{ (UTP)} + 0011 \text{ (NFP)} \\
    d & : 0111 \text{ (UTP)} + 0110 \text{ (NFP)}
\end{align*}
\]
Section 3.6.No 6

- Give a counterexample to show that CUTPNFP does not subsume UTPC. Using \( f = ab + cd \).

Notice that any NFP of \( f \) will be one of those red boxes.

Let’s take this as the DNF of \( \neg f \):

\[
\neg f = ac + ad + bc + bd
\]

We need to find a test-set, that satisfies CUTNFP on \( f \), but fails to give CUP coverage on \( f + \neg f \).

For example, dark blue + dark red boxes above give CUTPNFP on \( f \). By definition, this automatically gives CUP on \( f \); but not necessarily on \( \neg f \).

Now notice that the yellow boxes are the UTPs of \( \neg f \) that we need to cover.

The previous test-set does not indeed cover all those yellow boxes.