Ben Stanley (201603759)

Advanced Software Engineering - ACW

1)

```
int k,y;

...                          <-------- k0 y0
y=y-4*k-18;
                             <-------- k1 y1
while (k>3) {
      k--;
      y=y+4;
}                            <-------- y<0
```

Figure 1: Code Fragment

i)

Regarding the fragment of code above, the values of k and y at the respective stages must be calculated so that they cause the unsafe condition of "Y < 0" after the while statement. The cases of (K>3 == true) and (K>3 == false) must be examined (the while statement being executed n times versus the while loop not being executed).

With the case of the while loop not being executed, the K_1 value must be <=3 just before the while statement. For the Y_1 value, we are considering the unsafe condition to have been met, and the while statement has not executed, so we can take y as being less than 0 at Y_1.

Therefore, the K_1 and Y_1 values for the case (K>3 == false) are as follows:

$K_1 <= 3$

$Y_1 < 0$

Regarding the while statement being executed n times, we can first take the value of K_1>3, thus causing the while statement to be executed at least once. Furthermore, after the n^{th} iteration, when the statement has been executed, we can conclude the following three statements:

[1] $Y_n = K_1 + 4_n$ (because Y increases by 4 with every iteration)

[2] $K_n = K_1 - n$ (because K decreases by 1 with every iteration), so therefore

$n = K_1 - K_n$

[3] $K_n = 3$ (because as the loop has reached its exit point)

Using the above statements, we can calculate by substitution into the following formula:

$Y_n = Y_1 + 4(K_1 - K_n)$

Fill in Kn=3

$Y_n = Y_1 + 4(K_1 - 3)$

Expand brackets

$Y_n = Y_1 + 4K_1 - 12$

Yn must be < 0 to cause the unsafe condition

$Y_1 + 4K_1 - 12 < 0$

Rearrange

$Y_1 + 4K_1 < 12$

So therefore, for the case of (K>3 == true), we have the following conditions for K_1 and Y_1:

$K_1 > 3$

$Y_1 + 4K_1 < 12$

ii)

The K_0 and Y_0 values are the values of K and Y before the assignment "Y=Y–4*K–18". With the four above values we have from part i) being K_1 and Y_1 values for both (K>3==true) and (K>3==false), we can then work backwards and calculate the respective K_0 and Y_0 values for these. These being the conditions that K and Y must meet in the higher lines of code to provoke the unsafe condition Y<0 at the end of the code fragment.

For (K>3==false), the K value is not changed during the assignment of y, so we can conclude that K remains constant between K1 and K0. Thus, in this case as K_1<=3, K_0<=3. For the Y_0 value, substitute Y_1<0 into the equation as follows:

$Y_1 = Y_0 - 4K_0 - 18$

$Y_0 - 4K_0 - 18 < 0$

$Y_0 - 4K_0 < 18$ (Cannot be simplified further)

So, for the K_0 and Y_0 values of the case (K>3==false), to meet the unsafe condition Y<0, we have the following:

$K_0 <= 3$

$Y_0 - 4K_0 < 18$

Regarding the case of (K>3==true), as seen above the K value remains the same. Therefore, to cause the unsafe condition of Y<0, we can conclude that if K_1>3, then K_0>3.

For the Y_0 value in this case, we can do the following rearranging and substitution:

$Y_1 + 4K_1 < 12$

Rearrange to make Y_1 the subject

$Y_1 < 12 - 4K_1$

We know that $Y_1 = Y_0 - 4K_0 - 18$ so substitute in

$Y_0 - 4K_0 - 18 < 12 - 4K_1$

We know that $K_0 == K_1$, so these cancel out

$Y_0 - 18 < 12$

Rearrange to make Y_0 the subject

$Y_0 < 30$

In conclusion, for the K_0 and Y_0 values for the while statement to be executed n times (K>3==true) with the unsafe condition (Y<0) being met, we have the following:

$K_0 > 3$

$Y_0 < 30$

iii)

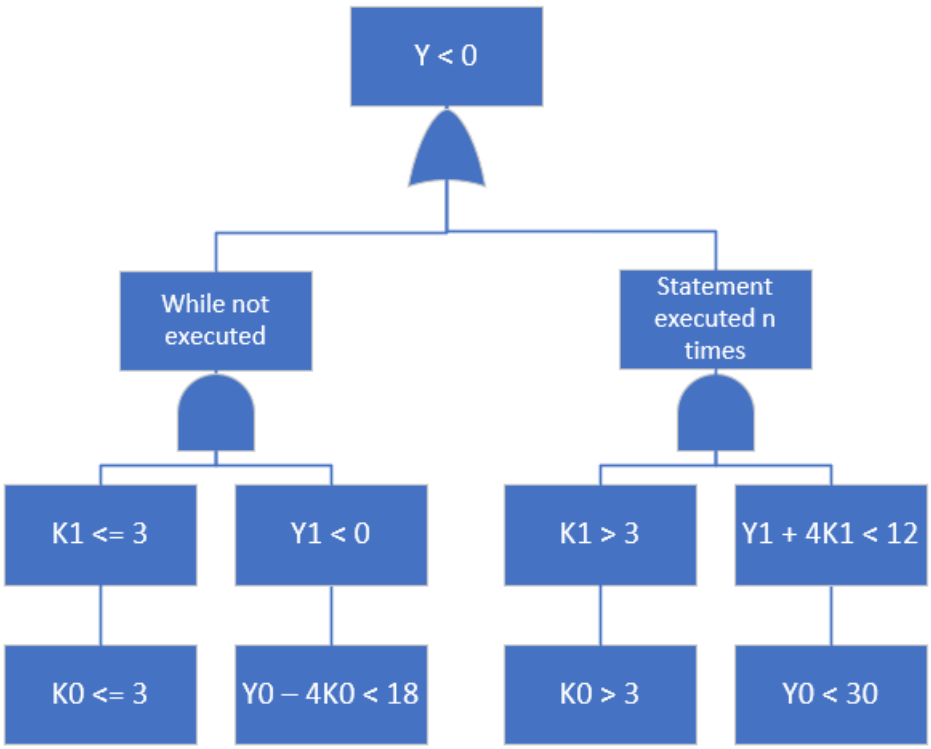

Figure 2: Fault Tree Analysis for unsafe condition Y<0

2)

i)

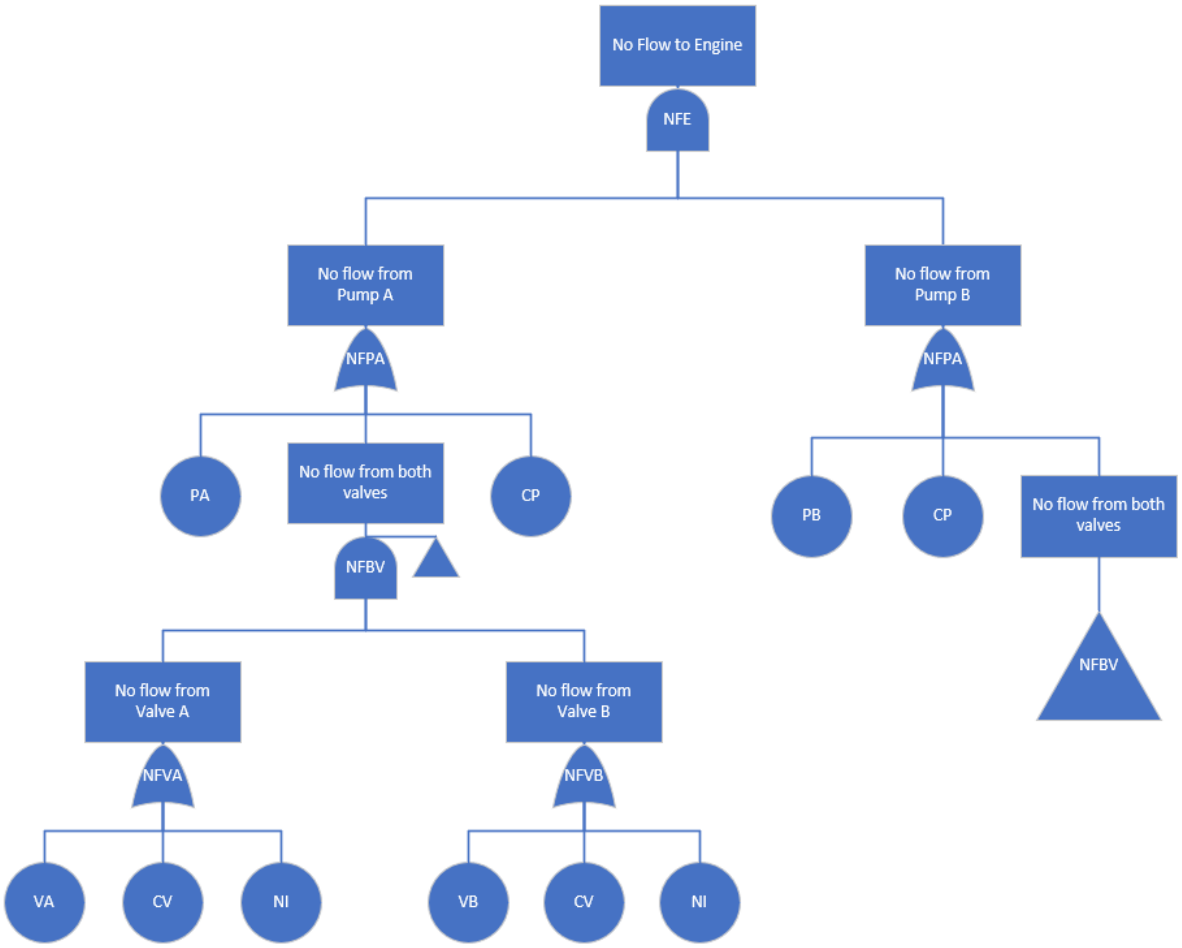

Figure 3: Fault Tree Analysis for No flow to Engine

ii)

The algorithm for obtaining minimal cut sets requires a number of steps, which first involves translating the structure of the tree into a parenthesised logical expression, containing only basic events.

NFE = NFPA . NFPB

= (CP + PA + NFBV) . (CP + PB + NFBV)

= (CP + PA + NFVA.NFVB) . (CP + PB + NFVA.NFVB)

= (CP + PA + (VA+VC+NI).(VB+CV+NI)) . (CP + PB + (VA+VC+NI).(VB+CV+NI))

The next step is to remove the parentheses using the distributive rule in Boolean logic.

= (CP + PA + VA.VB + VA.VC + VA.NI + CV.VB + CV.CV + CV.NI + NI.VB + NI.CV + NI.NI) . (CP + PB + VA.VB + VA.VC + VA.NI + CV.VB + CV.CV + CV.NI + NI.VB + NI.CV + NI.NI)

As this statement is significantly expanded, we can apply the following three rules of Boolean logic to cut down as follows.

X.X = X	[1] repeated events
X+X = X	[2] repeated cut sets
X+X.Y = X.(1+Y) = X.1 = X	[3] non-minimal cut sets

After following these steps, we have the following:

(CP + PA + VA.VB + CV + NI) **.** (CP + PB + VA.VB + CV + NI)

We then can use the distributive rule once more, and as PA and PB are the only non-repeated cut sets between the parentheses, we remove PA, PB and the duplicates, and add PA.PB. This leaves us with the following, which is the group of minimal cut sets that cause NFE:

CP + PA.PB + VA.VB + CV + NI

iii)

Single points of failure within a system can be identified as minimal cut sets which have an order of one. These are critical failure events that by themselves cause the top event. For this system, the single points of failure are the following:

CP, CV, NI

iv)

One measure that could be incorporated into the controller to improve the design of this system could be the automated status monitoring of both the pumps and valves. This could determine if the pump/valve was currently working/open. The real time monitoring could be used to prevent further damage given symptoms of failures / abnormal behaviours being shown.

As the basic events CP and CV are also single points of failure (the controller inadvertently stopping either both pumps or both valves). A mechanism could be introduced so that just a single pump or valve could be closed, and the flow re-routed to the remaining open pathway. As stated in the system description, fuel is transferred to the engine when at least one valve is open, and one pump is in operation, thus the system may still operate.