

# Software Testing, Quality Assurance & Maintenance (ECE453/CS447/SE465): Assignment 2 Solutions

Patrick Lam

## Question 2 (12 points)

### Predicate 1

$$a \wedge (b \leftrightarrow c)$$

Determination analysis:  $a$  determines  $p$  iff

$$\begin{aligned} [\text{true} \wedge (b \leftrightarrow c)] \oplus [\text{false} \wedge (b \leftrightarrow c)] \\ (b \leftrightarrow c) \oplus \text{false} \\ (b \leftrightarrow c) \end{aligned}$$

i.e.  $b : \text{false}, c : \text{false}$  or  $b : \text{true}, c : \text{true}$ .

$b$  determines  $p$  iff

$$\begin{aligned} [a \wedge (\text{true} \leftrightarrow c)] \oplus [a \wedge (\text{false} \leftrightarrow c)] \\ a \wedge c \oplus a \wedge \neg c \end{aligned}$$

i.e.  $a \wedge c = \text{true}$  and  $a \wedge \neg c = \text{false}$ , so  $a : \text{true}, c : \text{true}$ ; or  $a \wedge c = \text{false}$  and  $a \wedge \neg c = \text{true}$ , so  $a : \text{true}, c : \text{false}$ .

$c$  determines  $p$ : symmetric to  $b$ , need  $a : \text{true}, b : \text{true}$  or  $a : \text{true}, b : \text{false}$ .

Here is the truth table:

	$a$	$\wedge$	$(b \leftrightarrow c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true	true	✓	✓	✓
2	true		true	false		✓	✓
3	true		false	false		✓	✓
4	true		false	true	✓	✓	✓
5	false		true	false	✓		
6	false		true	false			
7	false		false	false			
8	false		false	false	✓		

GACC pairs: for  $a$ , we need one from  $\{1, 4\}$  and one from  $\{5, 8\}$ ; for  $b$ , we need one from  $\{1, 2\}$  and one from  $\{3, 4\}$ , and for  $c$  we need one from  $\{1, 3\}$  and one from  $\{2, 4\}$ .

CACC pairs: same as GACC for  $a$ ; for  $b$ , possibilities are  $(1, 3), (2, 4)$ ; for  $c$ , possibilities are  $(1, 2)$  and  $(3, 4)$ .

RACC pairs: for  $a$ , have  $(1, 5)$  and  $(4, 8)$ ; for  $b$ , have  $(1, 3), (2, 4)$ ; for  $c$ , have  $(1, 2)$  and  $(3, 4)$ .

GICC and hence RICC are infeasible for all clauses:  $p$  is always **false** when the major clause does not determine the predicate.

## Predicate 2

$$a \vee (b \wedge c) \vee d$$

Determination analysis: by using the textbook result on  $a \vee *$ , we see that  $a$  determines  $p$  iff  $(b \wedge c) \vee d$  is false; that is,  $d : \text{false}$  and at least one of  $b, c$  are false, so the possibilities are  $\langle b : \text{false}, c : \text{true}, d : \text{false} \rangle$ ;  $\langle b : \text{true}, c : \text{false}, d : \text{false} \rangle$ ; or  $\langle b : \text{true}, c : \text{true}, d : \text{false} \rangle$ . Symmetrically,  $d$  determines  $p$  when  $\langle a : \text{false}, b : \text{false}, c : \text{true} \rangle$ ;  $\langle a : \text{false}, b : \text{true}, c : \text{false} \rangle$ ; or  $\langle a : \text{false}, b : \text{true}, c : \text{true} \rangle$ .

For  $b$ , we carry out the analysis:

$$a \vee (b \wedge c) \vee d \oplus a \vee (\text{false} \wedge c) \vee d$$

$$(a \vee d) \vee c \oplus (a \vee d)$$

so we need  $a \vee d$  true,  $a \vee d \vee c$  false, which is impossible; or  $a \vee d$  false,  $a \vee d \vee c$  true, giving  $\langle a : \text{false}, c : \text{true}, d : \text{false} \rangle$ . Symmetrically for  $c$  we need  $\langle a : \text{false}, b : \text{true}, d : \text{true} \rangle$ .

	$a$	$\vee$	$(b$	$\wedge$	$c)$	$\vee$	$d$	$p$	$a$ det?	$b$ det?	$c$ det?	$d$ det?
1	true		true		true		true	true				
2	true		true		true		false	true				
3	true		true		false		true	true				
4	true		true		false		false	true	✓			
5	true		false		true		true	true				
6	true		false		true		false	true	✓			
7	true		false		false		true	true				
8	true		false		false		false	true	✓			
9	false		true		true		true	true				
10	false		true		true		false	true		✓	✓	
11	false		true		false		true	true				✓
12	false		true		false		false	false	✓		✓	✓
13	false		false		true		true	true				✓
14	false		false		true		false	false	✓	✓		✓
15	false		false		false		true	true				✓
16	false		false		false		false	false	✓			✓

GACC: for  $a$ , need one of  $\{4, 6, 8\}$  and one of  $\{12, 14, 16\}$ . For  $b$ , must use  $\langle 10, 14 \rangle$ . For  $c$ , must use  $\langle 10, 12 \rangle$ . For  $d$ , need one of  $\{11, 13, 15\}$  and one of  $\{12, 14, 16\}$ .

CACC: same as GACC in this case (by inspection).

RACC: for  $a$ , pairs  $\langle 4, 12 \rangle, \langle 6, 14 \rangle, \langle 8, 16 \rangle$ ; for  $b$  and  $c$ , same as CACC; for  $d$ , pairs  $\langle 11, 12 \rangle, \langle 13, 14 \rangle, \langle 15, 16 \rangle$ .

GICC: for  $a$ , infeasible, since when  $a$  does not determine  $p$ ,  $p$  is always true.

GICC is feasible for  $b$ ; one from each of:

$$b : \text{true}, p : \text{true} \quad \{1, 2, 3, 4, 9, 11\}$$

$b : \text{false}, p : \text{false}$	$\{14, 16\}$
$b : \text{true}, p : \text{false}$	12
$b : \text{false}, p : \text{true}$	$\{5, 6, 7, 8, 13, 16\}$

RICC for  $b$ :  $\langle 12, 16, X, Y \rangle$  where  $X, Y$  are one of  $\langle 1, 5 \rangle, \langle 2, 6 \rangle, \langle 3, 7 \rangle, \langle 4, 8 \rangle, \langle 9, 13 \rangle, \langle 11, 16 \rangle$ .

GICC for  $c$ : one from each of:

$c : \text{true}, p : \text{true}$	$\{1, 2, 5, 6, 9, 13\}$
$c : \text{false}, p : \text{false}$	$\{12, 16\}$
$c : \text{true}, p : \text{false}$	14
$c : \text{false}, p : \text{true}$	$\{3, 4, 7, 8, 11, 15\}$

RICC for  $c$ :  $\langle 14, 16, X, Y \rangle$  where  $X, Y$  are one of  $\langle 1, 3 \rangle, \langle 2, 4 \rangle, \langle 5, 7 \rangle, \langle 6, 8 \rangle, \langle 9, 11 \rangle, \langle 13, 15 \rangle$ .

GICC for  $d$  is infeasible: never have  $d$  not determining and  $p : \text{false}$ .

### Predicate 3

$$a \oplus b$$

$a$  and  $b$  determine  $p$  always (so the ICC are infeasible).

	$a$	$\oplus$	$b$	$p$	$a$ det?	$b$ det?
1	true		true	false	✓	✓
2	true		false	true	✓	✓
3	false		true	true	✓	✓
4	false		false	false	✓	✓

GACC: for  $a$ , need one of  $\{1, 2\}$  and one of  $\{3, 4\}$ ; for  $b$ , need one of  $\{1, 3\}$  and one of  $\{2, 4\}$ .

CACC and RACC: for  $a$ , pairs  $\langle 1, 3 \rangle$  and  $\langle 2, 4 \rangle$ ; for  $b$ , pairs  $\langle 1, 2 \rangle$  and  $\langle 3, 4 \rangle$ .

## Predicate 4

$$a \rightarrow (b \rightarrow c)$$

Determination analysis:  $a$  determines  $p$  iff

$$\text{true} \rightarrow (b \rightarrow c) \oplus \text{false} \rightarrow (b \rightarrow c)$$

$$b \rightarrow c \oplus \text{true} \\ \neg(b \rightarrow c)$$

so  $b : \text{true}, c : \text{false}$ .

$b$  determines  $p$  iff

$$a \rightarrow (\text{true} \rightarrow c) \oplus a \rightarrow (\text{false} \rightarrow c) \\ a \rightarrow c \oplus \text{true} \\ \neg(a \rightarrow c)$$

so  $a : \text{true}, c : \text{false}$ .

$c$  determines  $p$  iff

$$a \rightarrow (b \rightarrow \text{true}) \oplus a \rightarrow (b \rightarrow \text{false}) \\ a \rightarrow \text{true} \oplus a \rightarrow \neg b \\ \text{true} \oplus a \rightarrow \neg b \\ \neg(a \rightarrow \neg b)$$

so  $a : \text{true}, b : \text{true}$ .

	$a$	$\rightarrow$	$(b$	$\rightarrow$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true			✓
2	true		true		false	false	✓	✓	✓
3	true		false		true	true			
4	true		false		false	true		✓	
5	false		true		true	true			
6	false		true		false	true	✓		
7	false		false		true	true			
8	false		false		false	true			

GACC, CACC, RACC:  $a: \langle 2, 6 \rangle$ ;  $b: \langle 2, 4 \rangle$ ;  $c: \langle 1, 2 \rangle$ .

GICC and RICC: infeasible: for  $a$ , no  $a : \text{false}, p : \text{false}$  case; for  $b$ , no  $b : \text{false}, p : \text{false}$  case; for  $c$ , no  $c : \text{true}, p : \text{false}$  case.

## Predicate 5

$$a \leftrightarrow (b \vee c)$$

Determination analysis:  $a$  determines  $p$  always ( $\leftrightarrow$ );  $b$  determines  $p$  iff

$$\begin{aligned} a \leftrightarrow (\text{true} \vee c) &\oplus a \leftrightarrow (\text{false} \vee c) \\ a \leftrightarrow \text{true} &\oplus a \leftrightarrow (\text{false} \vee c) \\ a &\oplus a \leftrightarrow (\text{false} \vee c) \end{aligned}$$

i.e.  $a : \text{true}, a \leftrightarrow c : \text{false}$ , so  $a : \text{true}, c : \text{false}$ ; and  $a : \text{false}, a \leftrightarrow c : \text{true}$ , so  $a : \text{false}, c : \text{false}$ .

Symmetrically for  $c$ : need  $a : \text{true}, b : \text{false}$  or  $a : \text{false}, b : \text{false}$ .

	$a$	$\rightarrow$	$(b$	$\rightarrow$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true	✓		
2	true		true		false	true	✓	✓	
3	true		false		true	true	✓		✓
4	true		false		false	false	✓	✓	✓
5	false		true		true	false	✓		
6	false		true		false	false	✓	✓	
7	false		false		true	false	✓		✓
8	false		false		false	true	✓	✓	✓

For GACC for  $a$ , we need one of  $\{1, 2, 3, 4\}$  and one of  $\{5, 6, 7, 8\}$ . For  $b$ , we need one of  $\{2, 6\}$  and one of  $\{4, 8\}$ . For  $c$ , we need one of  $\{3, 7\}$  and one of  $\{4, 8\}$ .

For CACC, we need to make sure that  $p$  takes on both values; for  $a$ , that forces us to use one of the pairs  $\langle 1, 5 \rangle$ ,  $\langle 1, 6 \rangle$ ,  $\langle 1, 7 \rangle$ ,  $\langle 2, 5 \rangle$ ,  $\langle 2, 6 \rangle$ ,  $\langle 2, 7 \rangle$ ,  $\langle 3, 5 \rangle$ ,  $\langle 3, 6 \rangle$ ,  $\langle 3, 7 \rangle$ , and  $\langle 4, 8 \rangle$ .

For  $b$ , we have the pairs  $\langle 2, 4$  and  $\langle 6, 8$ , and for  $c$ , we have the pairs  $\langle 3, 4$  and  $\langle 7, 8$ .

To achieve RACC for  $a$ , we can use the pairs  $\langle 1, 5$ ,  $\langle 2, 6$ ,  $\langle 3, 7$ , and  $\langle 4, 8$ .

GICC and RICC are infeasible for  $a$ , since  $a$  always determines  $p$ . For  $b$  and  $c$ , they are feasible; for  $b$  we can use the 4-tuple  $\langle 1, 5, 3, 7$  and for  $c$  we can use the 4-tuple  $\langle 1, 5, 2, 6$ .

### Question 3 (13 points)

*For the following predicates and test sets:*

1.  $p = a \wedge (c \rightarrow b)$
2.  $p = a \vee (b \wedge c)$
3.  $p = a \oplus (b \wedge \neg c)$
4.  $p = a \leftrightarrow (b \rightarrow c)$
5.  $p = a \rightarrow (b \wedge c)$

$$T_1 = \{TTF, TFT, FTT, FTF, FFF\}$$

$$T_2 = \{TTT, TFF, FTT, FTF, FFT\}$$

$$T_3 = \{TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF\}$$

*Which coverage criteria do each of the test sets satisfy for each of the predicates?*

We can address CC and CoC immediately. Note that  $T_3$  achieves CoC and that  $T_1$  and  $T_2$  do not. Also, all of  $T_1, T_2$  and  $T_3$  achieve CC.

With some simple calculations (not shown), you can observe that  $T_1$  and  $T_2$  achieve PC on all predicates (and therefore so does  $T_3$ ).

We next address the active clause coverage criteria on a predicate-by-predicate basis. I found that the easiest way to do this was to create truth tables and carry out the determination analysis by eyeballing these truth tables (flip the major clause and see if  $p$  changes).

### Predicate 1

$$p = a \wedge (c \rightarrow b)$$

	$a$	$\wedge$	$(c$	$\rightarrow$	$b)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true	✓	✓	
2	true		false		true	true	✓		
3	true		true		false	false		✓	✓
4	true		false		false	true	✓		✓
5	false		true		true	false	✓		
6	false		false		true	false	✓		
7	false		true		false	false			
8	false		false		false	false	✓		

First, we observe that GACC, CACC and RACC are feasible, since each clause determines the predicate sometimes. Hence  $T_3$  achieves all of the active clause criteria. On the other hand, GICC (and hence RICC) are infeasible, because  $p$  is always false when  $a$  does not determine  $p$ .

We can observe that  $T_1$  achieves GACC, RACC and CACC for  $a$  but not for  $b$  or  $c$ , so  $T_1$  does not achieve GACC, RACC or CACC in general. As for  $T_2$ , we see that we do not achieve GACC with respect to  $b$  since it does not contain  $TFT$ , so  $T_2$  satisfies none of the active clause coverage criteria.

### Predicate 2

$$p = a \vee (b \wedge c)$$

	$a$	$\vee$	$(b$	$\wedge$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true			
2	true		true		false	true	✓		
3	true		false		true	true	✓		
4	true		false		false	true	✓		
5	false		true		true	true		✓	✓
6	false		true		false	false	✓		✓
7	false		false		true	false	✓	✓	
8	false		false		false	false	✓		

Again, all of GACC, CACC and RACC are feasible, so  $T_3$  achieves them all. GICC and RICC are infeasible again;  $p$  is never false when  $a$  does not determine  $p$ .

Note that active clause coverage requires the inputs  $FTT$  and  $FFT$  for  $b$  and additionally  $FTF$  for  $c$ .  $T_1$  does not contain  $FTT$ , so it does not achieve GACC, CACC or RACC.  $T_2$  contains all of these inputs, as well as the pair  $TFF/FTF$ , so it does achieve GACC. We also observe that the pair  $TFF/FTF$  also enables it to achieve CACC, since this pair makes  $p$  take on both values. Finally, since  $T_2$  only contains  $TFF$  with  $a = \text{true}$ , and does not contain  $FFF$ , it does not satisfy RACC.

### Predicate 3

$$p = a \oplus (b \wedge \neg c)$$

	$a$	$\oplus$	$(b$	$\wedge \neg$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true	✓		✓
2	true		true		false	false	✓	✓	✓
3	true		false		true	true	✓		
4	true		false		false	true	✓	✓	
5	false		true		true	false	✓		✓
6	false		true		false	true	✓	✓	✓
7	false		false		true	false	✓		
8	false		false		false	false	✓	✓	

We again observe that GACC, CACC and RACC are feasible, so  $T_3$  achieves them all. GICC and RICC are infeasible since  $a$  always determines  $p$ .

To check active clause coverage for  $T_1$ , we pick out the pair  $TTF/FTT$  for  $a$ ;  $TTF/FFF$  for  $b$ ; and  $FTT/TTF$  for  $c$ , so  $T_1$  achieves GACC. We need to pick  $TTF/FTF$  to achieve CACC on  $a$ . Moving on to  $b$ , we see that we need one of  $TTF, FTF$  and one of  $TFF, FFF$  to achieve GACC in general on  $b$ , but that CACC needs  $TTF$  with  $TFF$  and  $FTF$  with  $FFF$ . We do have  $FTF/FFF$ . Finally, for  $c$  we need one of the pairs  $TTT/TTF$  or  $FTT/FTF$ . We do have  $FTT/FTF$ , so  $T_1$  achieves CACC. The set  $TTF/FTF, FTF/FFF, FTT/FTF$  also achieves RACC.

For  $T_2$ , we reuse the analysis. The pair  $TTT/FTT$  is good for GACC, CACC and RACC on  $a$ . We must use  $FTF/TFF$  for GACC on  $b$ , and  $TTT/FTF$  for GACC on  $c$ , so conclude that  $T_2$  achieves GACC. Since the pair  $FTF/TFF$  is the only one that works for GACC and it does not achieve CACC, conclude that  $T_2$  does not achieve CACC or RACC.

#### Predicate 4

$$p = a \leftrightarrow (b \rightarrow c)$$

	$a$	$\leftrightarrow$	$(b$	$\rightarrow$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true	✓		✓
2	true		true		false	false	✓	✓	✓
3	true		false		true	true	✓		
4	true		false		false	true	✓	✓	
5	false		true		true	false	✓		✓
6	false		true		false	true	✓	✓	✓
7	false		false		true	false	✓		
8	false		false		false	false	✓	✓	

It turns out that this predicate is equivalent to predicate 3, so the results would of course be the same.

## Predicate 5

$$p = a \rightarrow (b \wedge c)$$

	$a$	$\rightarrow$	$(b$	$\wedge$	$c)$	$p$	$a$ det?	$b$ det?	$c$ det?
1	true		true		true	true		✓	✓
2	true		true		false	false	✓		✓
3	true		false		true	false	✓	✓	
4	true		false		false	false	✓		
5	false		true		true	true			
6	false		true		false	true	✓		
7	false		false		true	true	✓		
8	false		false		false	true	✓		

For general feasibility, we observe that all of the active clause criteria are feasible, so  $T_3$  achieves them all, and that the inactive clause criteria are infeasible because  $p$  is always true when  $a$  does not determine  $p$ .

To determine whether  $T_1$  and  $T_2$  achieve the active clause criteria, we start with  $c$ . It requires the pair  $TTT/TTT$ , which is not in  $T_1$ , so we can conclude that  $T_1$  does not satisfy GACC, CACC or RACC. The same pair is not in  $T_2$  either, so  $T_2$  does not satisfy GACC, CACC or RACC.

## Question 5 (20 points)

There are four predicates in the `pat()` method of `TestPat`. This method takes two inputs, `subject` and `pattern`, both arrays of characters. Note that `patternLen` and `subjectLen` ought to be `final` variables, since they are only assigned to once. I abbreviate them `pL` and `sL` respectively.

Answers to this question are extremely easy to test with an instrumented version of `TestPat`. You'll find diffs for a `TestPat` tester at:

<http://www.patricklam.ca/stqam/files/testpat.instrumenter.diff>

The book does not explain how to deal with code that occurs in a loop. For logic coverage, any path that reaches the predicate in question is fine.

Here are the predicates:

- line 47: `isPat == false && iSub + pL - 1 < sL`
- line 49: `subject[iSub] == pattern[0]`
- line 53: `iPat < pL`
- line 55: `subject[iSub+iPat] != pattern[iPat]`

I next describe the conditions to ensure reachability for each of these predicates:

- line 47: true (this line is unavoidable)
- line 49: `pL-1 < sL`: note that `isPat == false` is inevitable on the first iteration; the second clause is reachable on the first iteration (`iSub == 0`) when `pL-1 < sL`, for instance on the input `s=['a']`, `p=[]`.
- line 53: `s[iSub] == p[0] && pL-1 < sL`, for instance on the input `s=['a']`, `p=['a']`.
- line 55: `s[iSub] == p[0] && pL-1 < sL && pL > 1`, for instance on the input `s=['a', 'b']`, `p=['a', 'b']`.

Here are some possible conditions for predicate coverage. We can meet PC with `iSub==0` (i.e. on the first iteration), which simplifies our life; we'll assume that below.

- line 47 true: need to pick `s` and `p` such that `pL-1 < sL`, for instance `s=['a', 'b']`, `p=['z']`, giving `pL=1`, `sL=2` (expected value -1).
- line 47 false: can never make `isPat` false on first iteration (possible on later iterations, but that's harder); can easily violate `pL-1 < sL`, e.g. with `s=['a']`, `p=['x', 'y', 'z']` (expected value -1)

- line 49 true: need `s[0] == p[0]`, e.g. `s=['a'],p=['a']` (expected value 0)
- line 49 false: need `s[0] != p[0]`, e.g. `s=['a'],p=['z']` (expected value -1)
- line 53 true: when `iPat=1`, need `p` to have more than 1 element; also need `s[0]==p[0]` and `s` to be at least as long as `p`, e.g. `s=['a', 'b'], p=['a', 'b']` (expected value 0)
- line 53 false: if we want this with `iPat=1`, need `p` to be of length 1; to ensure reachability, need `p[0]==s[0] && pL-1 < sL`, for instance `iPat=1` and `s=['a'],p=['a']` (expected value 0)
- line 55 true: need conditions for line 53 true and `p` should not match `s` at a position after the first, e.g. `iPat=1, s=['a', 'b'],p=['a', 'c']` (expected value -1)
- line 55 false: need conditions for line 53 true and `p` should match `s`, for instance `iPat=1, s=['a', 'b'],p=['a', 'b']` (expected value 0)

Note that inspecting these cases suggests that making `pattern` a zero-length array will crash the method; you can test this and observe the crash. This is poor programming practice!

Mercifully, we've also ensured CC, CoC and CACC for the single-clause predicates at lines 49, 53 and 55; it remains to ensure coverage for the predicate at line 47, which has two clauses: `isPat == false` and `iSub + pL - 1 < sL`.

We have two clauses at line 47: `!isPat` and `iSub+pL-1<sL`. We'll need to vary `iSub` to cover these clauses.

- `!isPat` true: All inputs make `!isPat` true on the first iteration, for instance `s=['a'], p=['a']` (expected value 0).
- `!isPat` false: trickier: we must reach 53, make it false, and then wrap around the loop (so, see line 53 false from above). We can thus use the same input from above `s=['a'], p=['a']` (expected value 0), which makes `!isPat` false on the second input.

- $iSub + pL - 1 < sL$  true: again, the input  $s=['a']$ ,  $p=['a']$  works; all we need is the correct relationship between  $pL$  and  $sL$  on the first iteration.
- $iSub + pL - 1 < sL$  false: indeed, the input  $s=['a']$ ,  $p=['a']$  works again when  $iSub$  is 1; an alternative is  $s=['a']$ ,  $p=['b']$  (expected output -1); finally, any input where  $p$  is longer than  $s$  (e.g.  $s=['a']$ ,  $p=['b', 'b', 'b']$ , expected value -1) will cause this clause to be false on the first iteration.

Finally, for combinatorial coverage, we have four cases: TT, TF, FT, and FF.

- TT: boils down to making  $iSub + pL - 1 < sL$  true, since  $isPat$  is always false on the first iteration. Again,  $s=['a']$ ,  $p=['a']$  meets the length condition.
- TF: make  $pL$  greater than  $sL$ , e.g.  $s=['a']$ ,  $p=['a', 'b']$  (expected value -1).
- FT: only possible on second iteration; note that the second clause would never be evaluated due to short-circuit evaluation; however, make sure that  $s$  is long enough, and that  $p$  matches somewhere in  $s$ , e.g.  $s=['a', 'a']$ ,  $p=['a']$  (expected value 0).
- FF: make sure that  $p$  matches at the very end of  $s$ , e.g.  $s=['a', 'z']$ ,  $p=['z']$ ; there must be no more of  $s$  after we match  $p$ .

Since we've done CoC, and because CACC is feasible, we know that our CoC test case achieves CACC. Determination is very simple here, since we only have logical-and linking the clauses; we just need the minor clause to evaluate to T. The three first cases TT, TF, and FT from above are a test set that achieves CACC. (Note that CoC is not feasible for the `Triang` program in the book, due to relations between variables.)