**Solution to Homework 3**

Does criterion C1 subsume C2? That is, if a test set gives 100% coverage with respect to C1, will it also give 100% coverage with respect to C2?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | block | statement | condition | decision | CD | MC | c-use | p-use | all-use |
| block |  | Yes (1) | No (2) | Yes (3) | Yes (4) | Yes (5) | No (6) | Yes (7) | Yes (8) |
| statement | Yes (1) |  | No (2) | Yes (3) | Yes (4) | Yes (5) | No (6) | Yes (7) | Yes (8) |
| condition | No (2) | No (2) |  | No (9) | Yes (10) | Yes (11) | No (12) | No (13) | No (14) |
| decision | No (3) | No (3) | No (9) |  | Yes (15) | Yes (16) | No (17) | Yes (18) | Yes(19) |
| CD | No (4) | No (4) | No (10) | No (15) |  | Yes (20) | No (21) | No (22) | No (23) |
| MC | No (5) | No (5) | No (11) | No (16) | No (20) |  | No (24) | No (25) | No (26) |
| c-use | No (6) | No (6) | No (12) | No (17) | No (21) | No (24) |  | No (27) | Yes(28) |
| p-use | No (7) | No (7) | No (13) | No (18) | No (22) | No (25) | No (27) |  | Yes(29) |
| all-use | No (8) | No (8) | No (14) | No (19) | No (23) | No (26) | No (28) | No (29) |  |

1. block and statement

* 100% block coverage gives 100% statement coverage? 🡪 **Yes**

Use contradiction method

🡺 Make an assumption: a test suite gives 100% block coverage but not 100% statement coverage

🡺 This implies that every block is covered but some statements are still not covered

🡺 Some statements do not belong to any blocks

🡺 A contradiction is found 🡺 our assumption is not correct

🡺 a test suite that gives 100% block coverage must give 100% statement coverage

Every statement must belong to some block.

* 100% statement coverage gives 100% block coverage? 🡪 **Yes**

Use the similar contradiction method shown above

Same as above.

1. block/statement and condition

condition coverage = all the possibilities of every condition (i.e., T & F) in every decision needs to be covered

* 100% block/statement coverage gives 100% condition coverage? 🡪 **No**

...

\*\* conditions \*\*

**a b** T T  
 F F

if (a && b)

{

...

}

return 0;

Any test case that makes the if decision true can get 100% statement coverage...

but not 100% condition coverage.

* 100% condition coverage gives 100% block/statement coverage? 🡪 **No**

\*\* statement \*\*  
statement in the true branch was never executed

🡺 Not 100% block/statement coverage.

\*\* conditions \*\*  
 **a b**  
t1: T F  
t2: F T

all possibilities of each condition are covered 🡺 100% condition coverage

...

if (a && b)

{

...

}

return 0;

1. block/statement and decision

* 100% block/statement coverage gives 100% decision coverage? 🡪 **No**

...

if (a && b)

{

...

}

return 0;

A simple counter example: (T1: a = True, b = True), it gives 100% block/statement coverage. However, since the “False” outcome of the decision is not executed

🡺 not 100% decision coverage.

* 100% decision coverage gives 100% block/statement coverage? 🡪 **Yes**

A decision basically divides the execution path to 2 possible outcomes

A decision

A decision

With respect to each decision, if we covered its True and False outcomes

🡺 statements before this decision are covered

statements in the execution path of the True outcome are covered

statements in the execution path of the False outcome are covered

A statement can always be found in either one of the three locations:

1) before a decision

2) in the execution path of the True outcome of a decision

3) in the execution path of the False outcome of a decision

100% decision coverage means all three locations of a decision are covered regardless which decision is examined

a statement

“Statement” always in either one of the three locations

no matter which statement is examined, it is covered 🡺 100% statement coverage

1. block/Statement and condition-decision

* 100% block/statement coverage gives 100% condition-decision coverage? 🡪 **No**

Since 100% block/statement coverage cannot guarantee 100% condition or decision coverage, it cannot give 100% condition-decision coverage (which subsumes condition coverage and decision coverage).

* 100% condition-decision coverage gives 100% block/statement coverage? 🡪 **Yes**

Since 100% decision coverage gives 100% block/statement coverage, 100% condition-decision (which subsumes condition coverage and decision coverage) also gives 100% block/statement coverage.

1. block/Statement and multiple-condition

* 100% block/statement coverage gives 100% multiple-condition coverage? 🡪 **No**

*s*1: if (*a* > 0 && *b* > 0){

*s*2: *a*++;

*s*3: }else{

*s*4: *b*++;

*s*5: }

100% block cannot guarantee 100% condition

🡺 cannot give 100% multiple-condition coverage which is a stronger version of condition coverage

* 100% multiple-condition coverage gives 100% block/statement coverage? 🡪 **Yes**

Multiple-condition coverage requires every combination of simple condition to be tried once.

🡺 two outcomes of every decision to be covered 🡺 100% decision

🡺 100% block-statement coverage

1. block/statement and c-use

* 100% block/statement coverage gives 100% c-use coverage? 🡪 **No**

*s*1: if (*a* > 0) {

*s*2: *x* = 1

*s*3: }else{

*s*4: *x* = 2;

*s*5: }

*s*6: if (*b* > 0) {

*s*7: *y* = *x* + 1;

*s*8: }else{

*s*9: *y* = *x* – 1;

*s*10: }

The define/c-use pair of variable *x* contains (*s*2, *s*7), (*s*2, *s*9), (*s*4, *s*7), (*s*4, *s*9).

*t*1: {*a* = -1, *b* = 1} and *t*2: {*a* =1, *b* = -1} gives 100% block/statement coverage. However, only two pairs are covered (*s*2, *s*9), (*s*4, *s*7).

* 100% c-use coverage gives 100% block/statement coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, *s*5 is not covered.

1. block/statement and p-use

* 100% block/statement coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*x*);

*s*2: if (*x* > 0) {

*s*3: *y* = *x* + 1;

*s*4: }

*s*5: print(*y*);

*t*1: {*x* = 1} gives 100% block/statement coverage. 100% p-use coverage requires define/p-use pairs (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)) to be covered. However, only one of them, (*s*1, (*s*2, *s*3)) is covered.

* 100% p-use coverage gives 100% block/statement coverage? 🡪 **Yes**

*s*1: input (*x, y*);

*s*2: if (*x* > 0) {

*s*3: *x++*;

*s*4: }

*s*5: print(*x*);

*s*6: if (*y* > 0) {

*s*7: *y*++;

*s*8: }

*s*9: print(*y*);

100% p-use coverage guarantees that define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *x* are covered and define/p-use pairs, (*s*1, (*s*6, *s*7)) and (*s*1, (*s*6, *s*9)), of variable *y* are covered. It implies that all program branches are covered. Hence, all block/statements are also covered.

1. block/statement and all-use

* 100% block/statement coverage gives 100% all-use coverage? 🡪 **No**

Since 100% block/statement coverage does not give 100% c-use coverage or 100% p-use coverage, it cannot give 100% all-use coverage (which subsumes c-use and p-use coverage).

* 100% all-use coverage gives 100% block/statement coverage? 🡪 **Yes**

Since 100% p-use coverage give 100% block/statement coverage, 100% all-use coverage also gives 100% block/statement coverage.

1. condition and decision

* 100% condition coverage gives 100% decision coverage? 🡪 **No**

A simple counter example:

\*\* conditions \*\*  
t1: T F  
t2: F T

all possibilities of each condition are covered 🡺 100% condition coverage

\*\* decision \*\*  
only the false branch is covered

🡺 Not 100% decision coverage.

if (a && b)

* 100% decision coverage gives 100% condition coverage? 🡪 **No**

A simple counter example:

\*\* decision \*\*  
t1: T F 🡺 T  
t2: F F 🡺 F

🡺 100% decision coverage

\*\* condition \*\*  
T outcome of the condition “b” is not covered.

🡺 Not 100% condition coverage

if (a || b)

1. condition and condition-decision

* 100% condition coverage gives 100% condition-decision coverage? 🡪 **No**

Since 100% condition cannot give 100% decision coverage, it also cannot give 100% condition-decision coverage (which subsumes condition and decision coverage).

* 100% condition-decision coverage gives 100% condition coverage? 🡪 **Yes**

Condition-decision coverage subsumes condition coverage and decision coverage.

1. condition and multiple-condition

* 100% condition coverage gives 100% multiple-condition coverage? 🡪 **No**

Condition coverage cares about whether all the outcomes of each condition are EVENTUALLY covered, but it does not care HOW it is covered.

Multiple-condition coverage cares about not only “whether all the outcomes ... EVENTUALLY covered” but also “HOW ... are covered” 🡸 the definition specifically indicates every combination should be covered at least once.

A counter example:

if (a || b) {/\*option a\*/} else {/\*option b\*/}

a=false and b = true -> true

a= false and b = false -> false

The above statements will cover all outcomes of each condition but not every condition was covered at least once (AKA the coverage of every possible combination of conditions)

* 100% multiple-condition coverage gives 100% condition coverage? 🡪 **Yes**

If multiple-condition cares about EVENTUALLY and HOW, then EVENTUALLY is guaranteed.

🡺 100% condition coverage

if (a || b) {/\*option a\*/} else {/\*option b\*/}

100% MC will ensure that every combination of each condition will be covered, therefore,

every condition will be covered

1. condition and c-use

* 100% condition coverage gives 100% c-use coverage? 🡪 **No**

*s*1: input (*a*, *b*); *x* = 0; *y* = 0;

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *a* = *x* + 1;

*s*4: }else{

*s*5: *b* = *y* + 1;

*s*6: }

The define/c-use pair of variable *x* contains (*s*1, *s*3).

The define/c-use pair of variable *y* contains (*s*1, *s*5).

*t*1: {*a* = -1, *b* = 1} and *t*2: {*a* =1, *b* = -1} gives 100% condition coverage. However, the define/c-use pair of *y* is not covered.

* 100% c-use coverage gives 100% condition coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, *t*1 cannot make *a* > 0 as false.

1. condition and p-use

* 100% condition coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*a*, *b*);

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *x* ++;

*s*4: }else{

*s*5: *y* ++;

*s*6: }

100% p-use coverage requires define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *a* and define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, s5)), of variable *b* to be covered.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =1, *b* = -1} gives 100% condition coverage. However, only (*s*1, (*s*2, *s*3)) of *a* and (*s*1, (*s*2, *s*3)) of *b* are covered.

* 100% p-use coverage gives 100% condition coverage? 🡪 **No**

*s*1: input (*a*, *b*);

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *x* ++;

*s*4: }else{

*s*5: *y* ++;

*s*6: }

100% p-use coverage requires define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *a* and define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, s5)), of variable *b* to be covered.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =-1, *b* = -1} gives 100% p-use coverage. However, *a* > 0 cannot be true using these test cases.

1. condition and all-use

* 100% condition coverage gives 100% all-use coverage? 🡪 **No**

Since 100% condition cannot give 100% p-use or 100% c-use coverage, it also cannot give 100% all-use coverage (which subsumes c-use and p-use coverage).

* 100% all-use coverage gives 100% condition coverage? 🡪 **No**

*s*1: input (*a*, *b, x, y*);

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *x* ++;

*s*4: }else{

*s*5: *y* ++;

*s*6: }

100% p-use coverage requires define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *a* and define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, s5)), of variable *b* to be covered.

The define/c-use pair of variable *x* contains (*s*1, *s*3). The define/c-use pair of variable *y* contains (*s*1, *s*5).

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =-1, *b* = -1} gives 100% all-use (100% c-use and 100% p-use) coverage. However, *a* > 0 cannot be true using these test cases.

1. decision and condition-decision

* 100% decision coverage gives 100% condition-decision coverage? 🡪 **No**

Since 100% decision cannot give 100% condition coverage, it also cannot give 100% condition-decision coverage (which subsumes condition and decision coverage).

* 100% condition-decision coverage gives 100% condition coverage? 🡪 **Yes**

Condition-decision coverage subsumes condition coverage and decision coverage.

1. decision and multiple-condition

* 100% decision coverage gives 100% multiple-condition coverage? 🡪 **No**

Counter example 1:

if (a || b) {/\*option a\*/} else {/\*option b\*/}

a=true and b=true

a=false and b=false

The above statements will achieve 100% decision coverage for, but it doesn’t cover all the possible combinations of conditions.

Counter example 2:

*s*1: if (*a* > 0 || *b* > 0){

*s*2: *a*++;

*s*3: }else{

*s*4: *b*++;

*s*5: }

100% Multiple-condition coverage requires test cases which cover {*a* > 0 as true, *b* > 0 as true}, {*a* > 0 as true, *b* > 0 as false}, {*a* > 0 as false, *b* > 0 as true}, and {*a* > 0 as false, *b* > 0 as false}.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* = -1, *b* = -1} gives 100% decision coverage. However, they cannot cover {*a* > 0 as true, *b* > 0 as true} and {*a* > 0 as true, *b* > 0 as false}.

* 100% multiple-condition coverage gives 100% decision coverage? 🡪 **Yes**

Counter example 1:

if (a || b) {/\*option a\*/} else {/\*option b\*/}

If 100% MC coverage will get every combination of true/false at least once, then it has to

cover every decision path.

Counter example 2:

*s*1: if (*a* > 0 || *b* > 0){

*s*2: *a*++;

*s*3: }else{

*s*4: *b*++;

*s*5: }

100% Multiple-condition coverage requires test cases which cover {*a* > 0 as true, *b* > 0 as true}, {*a* > 0 as true, *b* > 0 as false}, {*a* > 0 as false, *b* > 0 as true}, and {*a* > 0 as false, *b* > 0 as false}. If 100% multiple-condition coverage is satisfied then all decision must be evaluated as both true and false.

1. decision and c-use

* 100% decision coverage gives 100% c-use coverage? 🡪 **No**

*s*1: if (*a* > 0) {

*s*2: *x* = 1

*s*3: }else{

*s*4: *x* = 2;

*s*5: }

*s*6: if (*b* > 0) {

*s*7: *y* = *x* + 1;

*s*8: }else{

*s*9: *y* = *x* – 1;

*s*10: }

The define/c-use pair of variable *x* contains (*s*2, *s*7), (*s*2, *s*9), (*s*4, *s*7), (*s*4, *s*9).

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =1, *b* = -1} gives 100% decision coverage. However, only two pairs are covered (*s*2, *s*9), (*s*4, *s*7).

* 100% c-use coverage gives 100% decision coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, *t*1 cannot make *a* > 0as false.

1. decision and p-use

* 100% decision coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*a*);

*s*2: if (*a* > 2) {

*s*3: *x* = *a \** 2;

*s*4: }else{

*s*5: *x* = *a +* 6;

*s*6: }

*s*7: if (*x* > 7) {

*s*8: *y* = *x* + 1;

*s*9: }else{

*s*10: *y* = *x* – 1;

*s*11: }

100% p-use coverage requires define/p-use pairs, (*s*3, (*s*7, *s*8)), (*s*3, (*s*7, *s*10)), (*s*5, (*s*7, *s*8)), and (*s*5, (*s*7, *s*10)) of variable *x* are covered.

*t*1: {*a* = 4}, *t*2: {*a* = 1} gives 100% decision coverage.

*t*1 covers (*s*3, (*s*7, *s*8)) of *x*.

*t*2 covers (*s*5, (*s*7, *s*10)) of *x*.

(*s*3, (*s*7, *s*10)) and (*s*5, (*s*7, *s*8)) are not covered.

* 100% p-use coverage gives 100% decision coverage? 🡪 **Yes**

*s*1: input (*a*);

*s*2: if (*a* > 2) {

*s*3: *x* = *a \** 2;

*s*4: }else{

*s*5: *x* = *a +* 6;

*s*6: }

*s*7: if (*x* > 7) {

*s*8: *y* = *x* + 1;

*s*9: }else{

*s*10: *y* = *x* – 1;

*s*11: }

100% p-use coverage requires define/p-use pairs, (*s*3, (*s*7, *s*8)), (*s*3, (*s*7, *s*10)), (*s*5, (*s*7, *s*8)), and (*s*5, (*s*7, *s*10)) of variable *x* are covered.

*t*1: {*a* = 4}, *t*2: {*a* =1}, *t*3: {*a* = 3}, *t*4: {*a* =2} gives 100% p-use coverage. They also guarantee 100% decision coverage.

Generally speaking, for p-use coverage, every p-use must independently contribute to the decision taking both a true and false branch. So it guarantees decision coverage.

1. decision and all-use

* 100% decision coverage gives 100% all-use coverage? 🡪 **No**

Since 100% decision coverage cannot give 100% p-use or 100% c-use coverage, it also cannot give 100% all-use coverage (which subsumes c-use and p-use coverage).

* 100% all-use coverage gives 100% decision coverage? 🡪 **Yes**

Since 100% p-use coverage gives 100% decision coverage, 100% all-use coverage (which subsumes p-use coverage) also gives 100% decision coverage.

1. condition-decision and multiple-condition

* 100% condition-decision coverage gives 100% multiple-condition coverage? 🡪 **No**

*s*1: if (*a* > 0 || *b* > 0){

*s*2: *a*++;

*s*3: }else{

*s*4: *b*++;

*s*5: }

100% Multiple-condition coverage requires test cases which cover

{*a* > 0 as true, *b* > 0 as true},

{*a* > 0 as true, *b* > 0 as false},

{*a* > 0 as false, *b* > 0 as true}, and

{*a* > 0 as false, *b* > 0 as false}.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* = -1, *b* = -1}, *t*3: {*a* = 1, *b* = -1} gives 100% condition-decision coverage.

However, they cannot cover {*a* > 0 as true, *b* > 0 as true}.

* 100% multiple-condition coverage gives 100% condition-decision coverage? 🡪 **Yes**

*s*1: if (*a* > 0 || *b* > 0){

*s*2: *a*++;

*s*3: }else{

*s*4: *b*++;

*s*5: }

100% Multiple-condition coverage requires test cases which cover

{*a* > 0 as true, *b* > 0 as true},

{*a* > 0 as true, *b* > 0 as false},

{*a* > 0 as false, *b* > 0 as true}, and

{*a* > 0 as false, *b* > 0 as false}.

If 100% multiple-condition coverage is satisfied, it implies all possible combinations of condition outcomes are covered so that 100% condition-decision coverage is achieved.

1. condition-decision and c-use

* 100% condition-decision coverage gives 100% c-use coverage? 🡪 **No**

*s*1: if (*a* > 0) {

*s*2: *x* = 1

*s*3: }else{

*s*4: *x* = 2;

*s*5: }

*s*6: if (*b* > 0) {

*s*7: *y* = *x* + 1;

*s*8: }else{

*s*9: *y* = *x* – 1;

*s*10: }

The define/c-use pair of variable *x* contains (*s*2, *s*7), (*s*2, *s*9), (*s*4, *s*7), (*s*4, *s*9).

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =1, *b* = -1} gives 100% condition-decision coverage. However, only two pairs are covered (*s*2, *s*9), (*s*4, *s*7).

* 100% c-use coverage gives 100% condition-decision coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, *t*1 cannot make *a* > 0as false.

1. condition-decision and p-use

* 100% condition-decision coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*a*);

*s*2: if (*a* > 2) {

*s*3: *x* = *a \** 2;

*s*4: }else{

*s*5: *x* = *a +* 6;

*s*6: }

*s*7: if (*x* > 7) {

*s*8: *y* = *x* + 1;

*s*9: }else{

*s*10: *y* = *x* – 1;

*s*11: }

100% p-use coverage requires define/p-use pairs, (*s*3, (*s*7, *s*8)), (*s*3, (*s*7, *s*10)), (*s*5, (*s*7, *s*8)), and (*s*5, (*s*7, *s*10)) of variable *x* are covered.

*t*1: {*a* = 4}, *t*2: {*a* = 1} gives 100% condition-decision coverage.

*t*1 covers (*s*3, (*s*7, *s*8)) of *x*.

*t*2 covers (*s*5, (*s*7, *s*10)) of *x*.

(*s*3, (*s*7, *s*10)) and (*s*5, (*s*7, *s*8)) are not covered.

* 100% p-use coverage gives 100% condition-decision coverage? 🡪 **No**

*s*1: input (*a*, *b*);

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *x* ++;

*s*4: }else{

*s*5: *y* ++;

*s*6: }

100% p-use coverage requires define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *a* and define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *b* are covered.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =-1, *b* = -1} gives 100% p-use coverage. However, *a* > 0 cannot be true using these test cases. So 100% condition-decision coverage is not satisfied.

1. condition-decision and all-use

* 100% condition-decision coverage gives 100% all-use coverage? 🡪 **No**

Since 100% condition-decision coverage cannot give 100% p-use or 100% c-use coverage, it also cannot give 100% all-use coverage (which subsumes c-use and p-use coverage).

* 100% all-use coverage gives 100% condition-decision coverage? 🡪 **No**

Since 100% all-use coverage cannot give 100% condition coverage, Hence 100% all-use coverage also cannot give 100% condition-decision coverage (which subsumes decision and condition coverage).

1. multiple-condition and c-use

* 100% multiple-condition coverage gives 100% c-use coverage? 🡪 **No**

*s*1: if (*a* > 0) {

*s*2: *x* = 1

*s*3: }else{

*s*4: *x* = 2;

*s*5: }

*s*6: if (*b* > 0) {

*s*7: *y* = *x* + 1;

*s*8: }else{

*s*9: *y* = *x* – 1;

*s*10: }

The define/c-use pair of variable *x* contains (*s*2, *s*7), (*s*2, *s*9), (*s*4, *s*7), (*s*4, *s*9).

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =1, *b* = -1} gives 100% multiple-condition coverage. However, only two pairs are covered (*s*2, *s*9), (*s*4, *s*7).

* 100% c-use coverage gives 100% multiple-condition coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, *t*1 cannot make *a* > 0as false.

1. multiple-condition and p-use

* 100% multiple-condition coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*a*);

*s*2: if (*a* > 2) {

*s*3: *x* = *a \** 2;

*s*4: }else{

*s*5: *x* = *a +* 6;

*s*6: }

*s*7: if (*x* > 7) {

*s*8: *y* = *x* + 1;

*s*9: }else{

*s*10: *y* = *x* – 1;

*s*11: }

100% p-use coverage requires define/p-use pairs, (*s*3, (*s*7, *s*8)), (*s*3, (*s*7, *s*10)), (*s*5, (*s*7, *s*8)), and (*s*5, (*s*7, *s*10)) of variable *x* are covered.

*t*1: {*a* = 4}, *t*2: {*a* = 1} gives 100% multiple-condition coverage.

*t*1 covers (*s*3, (*s*7, *s*8)) of *x*.

*t*2 covers (*s*5, (*s*7, *s*10)) of *x*.

(*s*3, (*s*7, *s*10)) and (*s*5, (*s*7, *s*8)) are not covered.

* 100% p-use coverage gives 100% multiple-condition coverage? 🡪 **No**

*s*1: input (*a*, *b*);

*s*2: if (*a* > 0 || *b* > 0){

*s*3: *x* ++;

*s*4: }else{

*s*5: *y* ++;

*s*6: }

100% p-use coverage requires define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *a* and define/p-use pairs, (*s*1, (*s*2, *s*3)) and (*s*1, (*s*2, *s*5)), of variable *b* are covered.

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =-1, *b* = -1} gives 100% p-use coverage. However, to satisfy 100% multiple condition coverage, test cases need to cover {*a* > 0 as true, *b* > 0 as true}, {*a* > 0 as true, *b* > 0 as false}, {*a* > 0 as false, *b* > 0 as true}, and {*a* > 0 as false, *b* > 0 as false}. Two test cases obviously cannot achieve such coverage.

1. multiple-condition and all-use

* 100% multiple-condition coverage gives 100% all-use coverage? 🡪 **No**

Since 100% multiple-condition coverage cannot give 100% p-use or 100% c-use coverage, it also cannot give 100% all-use coverage (which subsumes c-use and p-use coverage).

* 100% all-use coverage gives 100% multiple-condition coverage? 🡪 **No**

Since 100% all-use coverage cannot give 100% condition coverage, Hence 100% all-use coverage also cannot give 100% multiple-condition coverage.

1. c-use and p-use

* 100% c-use coverage gives 100% p-use coverage? 🡪 **No**

*s*1: input (*a, b*);

*s*2: if (*a* > 0){

*s*3: print (*a* + *b*);

*s*4: }else{

*s*5: print (“Hello World!”);

*s*6: }

*s*5 does not contain c-use of any variable. *t*1: {*a* = 1, *b* = 1} gives 100% c-use coverage. However, define/p-use pair of variable *a*, (*s*1, (*s*2, *s*5)), is not covered.

* 100% p-use coverage gives 100% c-use coverage? 🡪 **No**

*s*1: input (*a*, *b*)

*s*2: if (*a* > 0) {

*s*3: *x* = 1

*s*4: }else{

*s*5: *x* = 2;

*s*6: }

*s*7: if (*b* > 0) {

*s*8: *y* = *x* + 1;

*s*9: }else{

*s*10: *y* = *x* – 1;

*s*11: }

The define/c-use pair of variable *x* contains (*s*3, *s*8), (*s*3, *s*10), (*s*5, *s*8), (*s*5, *s*10).

The define/p-use pair of variable *a* contains (*s*1, (*s*2, *s*3)), (*s*1, (*s*2, *s*5))

The define/p-use pair of variable *b* contains (*s*1, (*s*7, *s*8)), (*s*1, (*s*7, *s*10))

*t*1: {*a* = -1, *b* = 1}, *t*2: {*a* =1, *b* = -1} gives 100% p-use coverage. However, two define/c-use pairs of *x* are not covered (*s*3, *s*8), (*s*5, *s*10).

1. c-use and all-use

* 100% c-use coverage gives 100% all-use coverage? 🡪 **No**

100% c-use does not give 100% p-use (which is subsumed by all-use).

* 100% all-use coverage gives 100% c-use coverage? 🡪 **Yes**

All-use subsumes c-use.

1. p-use and all-use

* 100% c-use coverage gives 100% all-use coverage? 🡪 **No**

100% p-use does not give 100% c-use (which is subsumed by all-use).

* 100% all-use coverage gives 100% c-use coverage? 🡪 **Yes**

All-use subsumes p-use.