Exercise 2: Static Taint Analysis

Consider the following program

```
L1 String a = getInput(); //getInput takes value from the  environment
L2 String b = "";
L3 b = append(b, "abc";
L4 b = append(b, a)
L5 String sink = b
L6 println(sink);   //println is a sensible function

L7 String append(String s1, String s2) {
L8    return s1 ^ s2;
      }
```

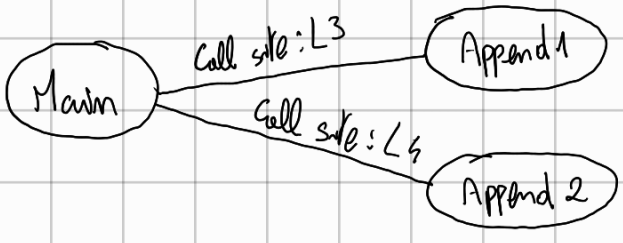
① • Which syntactic conditions the program above must fulfil in oder to apply static taint analysis? ?

② • Discuss how static taint analysis is applied to discover whether the previous program fragments presents taint flow. Specifically define and illustrate the constraints generated by the static taint analysis.

② Call site 1: L3. Context: { b → untainted, "abc" → untainted }, Taint In: No, Taint out: No

Call site 2: L4. Context: { a → tainted, b → untainted }, Taint In: Yes, Taint out: Yes

CALL GRAPH:



Analysis:

$OUT(S) = (IN(S) \setminus KILL(S)) \cup GEN(S)$

Label	GEN	KILL	IN	OUT
L1	{a}	∅	∅	{a}
L2	∅	{b}	{a}	{a}
L3	∅	{b}	{a}	{a}
L4	{b}	∅	{a}	{a,b}
L5	{sink}	∅	{a,b}	{a,b,sink}
L6	∅	∅	{a,b,sink}	{a,b,sink}

At program point 6 println(sink) receives a tainted input (sink). The program is not secure and cannot be executed.

Exercise 2

Consider the following WASM program

```
(func $test (param $x i32) (result i32)
 (local $i i32)
  local.get $x
  local.set $i
  (loop $countdown
    local.get $i
    i32.eqz
    br_if 1 ;; exit loop if i == 0
    local.get $i
    i32.const 1
    i32.sub
    local.set $i
    br $countdown
  )
  local.get $i
)
```

1. Simulate the execution of this function with actual argumnet 3.Show step-by-step stack, local variable state and the operational rule applied.
2. Explain the semantics of `loop`, `br`, and `br_if` in terms of the small-step operational rules.

For 1. you show configuration and how it evolves. Since we are executing this function, we will represent it's frame. We assume the instructions just executed were: i32.const 3, call 0.

Frame $C = \{$ module m, memory m^*_{mem}, locals $x=3; i=0$, stack ε, instr local.get \$x, $e^*\}$

for simplicity I will only represent frame showing what changed from last step.

$C = \{$ stack 3, instr local.set \$i, $e^*\}$
$C = \{$ stack ε, locals $x=3; i=3$, instr loop e^*_{body}, $e^*\}$

Given administrative encoding rule:

$C = \{$ instr label $\{$ loop $e^*_{body}\}$ $(\varepsilon$; local.get \$i, $e^*_1)$, $e^*_2\}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (3; \text{i32.eqz}, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (0; \text{br\_if} \ 1, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (\varepsilon; \text{local.get} \ \$v, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (3; \text{i32.const} \ 1, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (1,3; \text{i32.sub}, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (2; \text{local.set} \ \$v, e_1^*), e_2^* \}$

$c = \{ \text{locals} \ x=3; v=2, \ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (\varepsilon; \text{br countdown}), e_2^* \}$

Repeat this until $v=0$ and you get:

$c = \{ \text{locals} \ x=3; v=0, \ \text{vmstr} \ \text{label} \{ \text{loop} \ e^*_{\text{body}} \} (1; \text{br\_if} \ 1, e_1^*), e_2^* \}$

$c = \{ \text{vmstr} \ \text{local.get} \ \$v \}$

$c = \{ \text{module} \ m, \ \text{memory} \ m^*_{\text{mem}}, \ \text{locals} \ x=3; v=0, \ \text{stack} \ 0, \ \text{vmstr} \ \text{return} \}$

We will return value 0.

Semantics of rules:

loop:

$$\frac{-}{\{\text{instr loop } e^*_{body}\} \to \{\text{instr label } \{\text{loop } e^*_{body}\}(\varepsilon;\ e^*_{body})\}}$$

br s:

$$\frac{s > 0}{\{\text{instr label } \{e^*_{cont}\}(-;\ br\ s,\ -)\} \to \{\text{instr br } s-1\}}$$

br 0:

$$\frac{-}{\{\text{instr label } \{e^*_{cont}\}(-;\ br\ 0,\ -)\} \to \{\text{instr } e^*_{cont}\}}$$

br_if:

$$\frac{n \neq 0}{\{\text{stack } n;\ \text{instr br\_if } s\} \to \{\text{instr br } s\}}$$

br_if:

$$\frac{n = 0}{\{\text{stack } n;\ \text{instr br\_if } s\} \to \{\}}$$

Exercise 3

Consider an intermediate programming language designed to enable near-native code execution. We assume that the language includes standard instructions and is equipped with state information to tag run-time data with **taint** or **untaint** tags. WILL BE MODIFIED

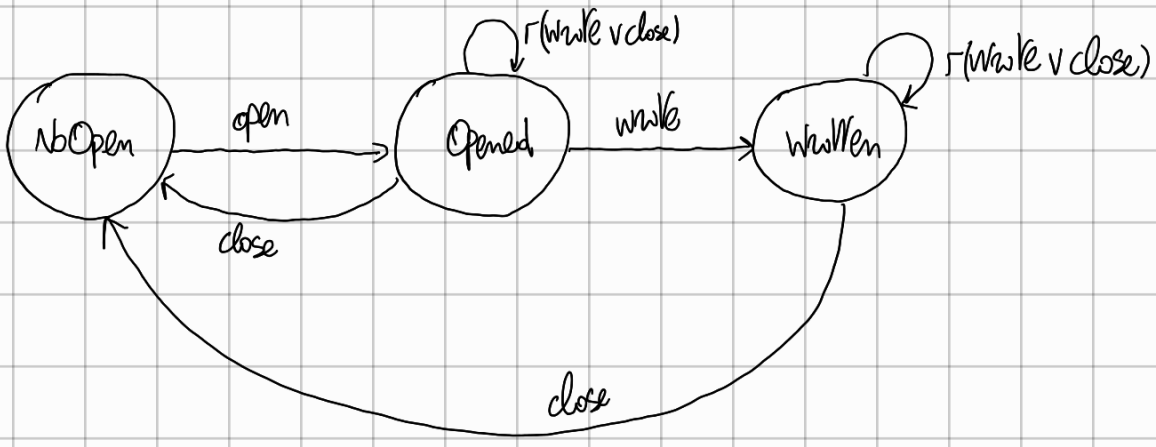
1. Design the *Security Automaton* for operating over files. A file must be opened before any operations. After it is opened, it can be written at most once, and then it can be read multiple times. It must be closed before being reopened.

2. Exploit the security automata defined in the previous question to instrument the code below

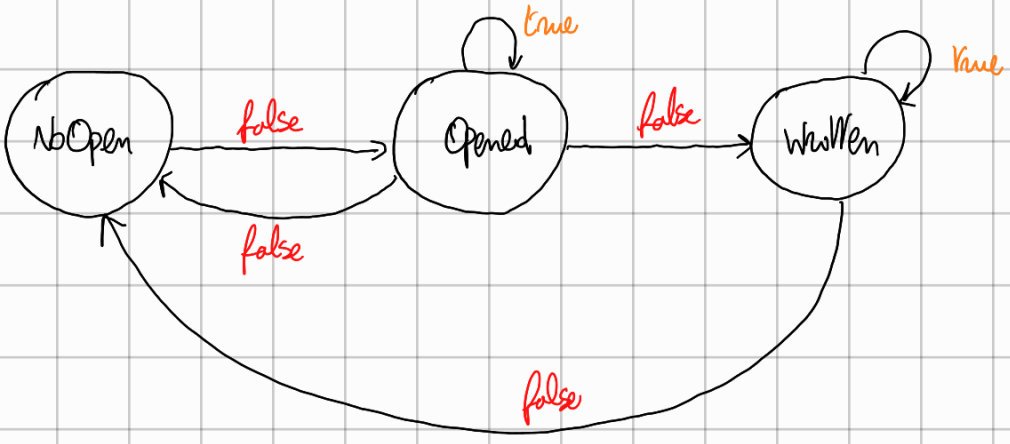
```
z=readf(f);
write(f, z1);
write(f, z2);
```

Note: file can be read as long as it is opened, but can only be written once. You can read before writing (solid in class).

FSA:

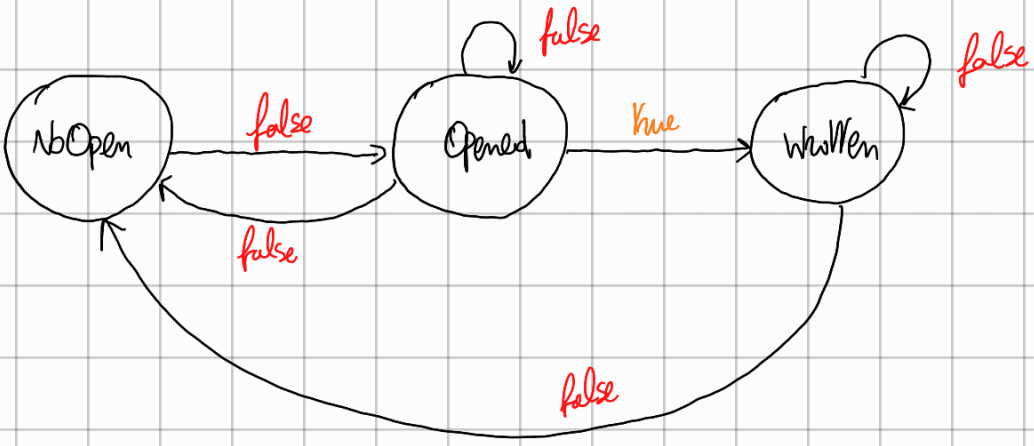


1. z = read(f);



instrumented code: if(!(state == Opened || state == Written)) abort;

2. write(f, z1);



if (state == Opened) : state = Written;
else abort;

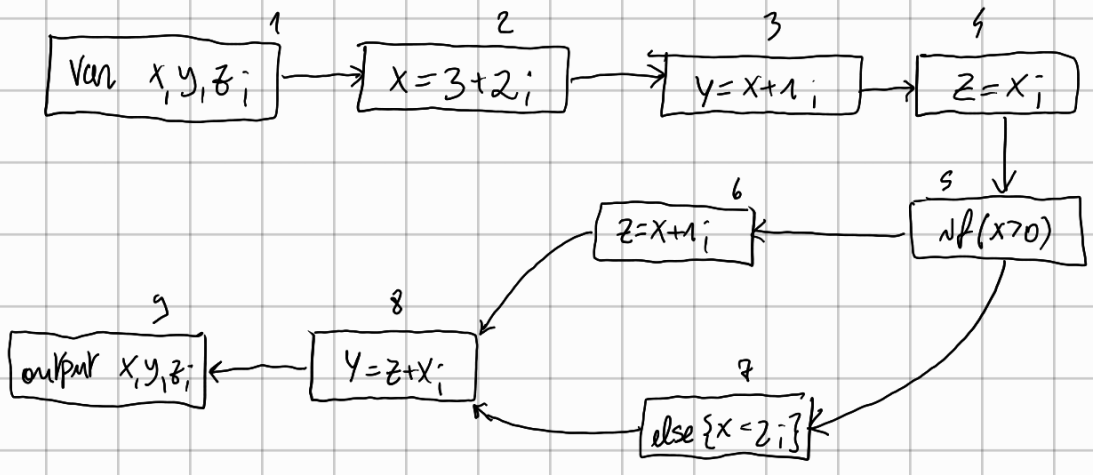
Same for write(f, z2);
Here you might directly abort because you already wrote (from classes, UNCLEAR)

Exercise 3: Liveness

Apply liveness analysis on the following piece of code

```
var  x,y,z;
x = 3 + 2;
y = x + 1;
z = x;
if (x > 0) {z = x + 1; } else {x = 2;}
y = z + x;
output  x,y,z;
```

- draw the corresponding control flow graph, and compute the constraints for each block,
- outline the least solution of the constraints.



$[entry] = \emptyset$

$[\text{var } x,y,z;] = [x=3+2;] \setminus \{x,y,z\}$

$[x=3+2;] = [y=x+1;] \setminus \{x\}$

$[y=x+1;] = [z=x;] \setminus \{y\} \cup \{x\}$

$[z=x;] = [\text{if}(x>0)] \setminus \{z\} \cup \{x\}$

$[\text{if}(x>0)] = ([z=x+1;] \cup [\text{else}\{x=2;\}]) \cup \{x\}$

$[z=x+1;] = ([y=z+x;] \setminus \{z\}) \cup \{x\}$

$[\text{else}\{x=2;\}] = [y=z+x;] \setminus \{x\}$

$[y=z+x;] = [\text{output } x,y,z;] \setminus \{y\} \cup \{x,z\}$

$[\text{output } x,y,z] = [exit] \cup \{x,y,z\}$

$[exit] = \emptyset$

It 0:

$[entry] = \emptyset$

$[var\ x,y,z;] = [x=3+z;] \setminus \{x,y,z\}$

$[x=3+2;] = [y=x+1;] \setminus \{x\}$

$[y=x+1;] = [z=x;] \setminus \{y\} \cup \{x\}$

$[z=x;] = [wf(x>0)] \setminus \{z\} \cup \{x\}$

$[wf(x>0)] = ([z=x+1;] \cup [else\{x=z;\}]) \cup \{x\}$

$[z=x+1;] = ([y=z+x;] \setminus \{z\}) \cup \{x\}$

$[else\{x=z;\}] = [y=z+x;] \setminus \{x\}$

$[y=z+x;] = [output\ x,y,z;] \setminus \{y\} \cup \{x,z\}$

$[output\ x,y,z] = [exit] \cup \{x,y,z\}$

$[exit] = \emptyset$

It 1:

$[entry] = \emptyset$

$[var\ x,y,z;] = \emptyset$

$[x=3+2;] = \emptyset$

$[y=x+1;] = \{x\}$

$[z=x;] = \{x\}$

$[wf(x>0)] = \{x,z\}$

$[z=x+1;] = \{x\}$

$[else\{x=z;\}] = \{z\}$

$[y=z+x;] = \{x,z\}$

$[output\ x,y,z] = \{x,y,z\}$

$[exit] = \emptyset$

We reached faster convergence by using values computed during current iteration to complete the current iteration itself. For example, for $[else\{x=z;\}]$ it was faster to use the $\{x,z\}$ computed for $[y=z+x;]$ instead of losing time and work with $\emptyset$.

Normal approach:

$[entry] = \emptyset$

$[var\ x,y,z;] = [x=3+z;] \setminus \{x,y,z\}$

$[x=3+2;] = [y=x+1;] \setminus \{x\}$

$[y=x+1;] = [z=x;] \setminus \{y\} \cup \{x\}$

$[z=x;] = [wf(x>0)] \setminus \{z\} \cup \{x\}$

$[wf(x>0)] = ([z=x+1;] \cup [else\{x=z;\}]) \cup \{x\}$

$[z=x+1;] = ([y=z+x;] \setminus \{z\}) \cup \{x\}$

$[else\{x=z;\}] = [y=z+x;] \setminus \{x\}$

$[y=z+x;] = [output\ x,y,z;] \setminus \{y\} \cup \{x,z\}$

$[output\ x,y,z] = [exit] \cup \{x,y,z\}$

$[exit] = \emptyset$

It 0:

$[entry] = \emptyset$

$[var\ x,y,z;] = \emptyset$

$[x=3+2;] = \emptyset$

$[y=x+1;] = \emptyset$

$[z=x;] = \emptyset$

$[wf(x>0)] = \emptyset$

$[z=x+1;] = \emptyset$

$[else\{x=z;\}] = \emptyset$

$[y=z+x;] = \emptyset$

$[output\ x,y,z] = \emptyset$

$[exit] = \emptyset$

Ir 1:

[entry] = ∅

[var x, y, z;] = ∅

[x = 3+2;] = ∅

[y = x+1;] = {x}

[z = x;] = {x}

[if (x>0)] = {x}

[z = x+1;] = {x}

[else {x=2;}] = ∅

[y = z+x;] = {x, z}

[output x, y, z] = {x, y, z}

[exit] = ∅

Ir 2:

[entry] = ∅

[var x, y, z;] = ∅

[x = 3+2;] = ∅

[y = x+1;] = {x}

[z = x;] = {x}

[if (x>0)] = {x}

[z = x+1;] = {x}

[else {x=2;}] = {z}

[y = z+x;] = {x, z}

[output x, y, z] = {x, y, z}

[exit] = ∅

Ir 3:

[entry] = ∅

[var x, y, z;] = ∅

[x = 3+2;] = ∅

[y = x+1;] = {x}

[z = x;] = {x}

[if (x>0)] = {x, z}

[z = x+1;] = {x}

[else {x=2;}] = {z}

[y = z+x;] = {x, z}

[output x, y, z] = {x, y, z}

[exit] = ∅

Set Theory Approach:

$IN(S) = GEN(S) \cup (OUT(S) \setminus KILL(S))$

$OUT(S) = \bigcup_{W \in SUCC(S)} IN(W)$

Prog. Points	GEN	KILL	OUT	IN
P.1	$\emptyset$	$\{x,y,z\}$	$\emptyset$	$\emptyset$
P.2	$\emptyset$	$\{x\}$	$\{x\}$	$\emptyset$
P.3	$\{x\}$	$\{y\}$	$\{x\}$	$\{x\}$
P.4	$\{x\}$	$\{z\}$	$\{x,z\}$	$\{x\}$
P.5 (if)	$\{x\}$	$\emptyset$	$\{x,z\}$	$\{x,z\}$
P.6	$\{x\}$	$\{z\}$	$\{x,z\}$	$\{x\}$
P.7	$\emptyset$	$\{x\}$	$\{x,z\}$	$\{z\}$
P.8 (Merge)	$\{z,x\}$	$\{y\}$	$\{x,y,z\}$	$\{x,z\}$
P.9	$\{x,y,z\}$	$\emptyset$	$\emptyset$	$\{x,y,z\}$

Exercise 1: Flow Types

Consider the following program

```
if (a == b) then {
        while (a >0) {a = a -1;}
  else {a = a -1;}
```

- Identify the information flow security policies over the lattice $L \leq H$ namely the typing context Γ, under which the program above is well typed by considering the rules of the static type system presented during the lectures.

$$\frac{\Gamma, \bot \vdash (a==b) : \ell \qquad \Gamma, \ell \vdash \text{while}(a>0)\{a=a-1;\} \qquad \Gamma, \ell \vdash a=a-1;}{\Gamma, \bot \vdash \text{if}\,(a==b)\,\text{then}\,\{\text{while}\,(a>0)\,\{a=a-1;\}\,\text{else}\,\{a=a-1;\}}$$

$$\frac{\Gamma \vdash a : \ell_a \qquad \Gamma \vdash b : \ell_b}{\Gamma \vdash (a==b) : \ell = \ell_a \sqcup \ell_b}$$

$$\frac{\Gamma \vdash (a>0) : \ell_a \qquad \Gamma, \ell \sqcup \ell_a \vdash a=a-1;}{\Gamma, \ell \vdash \text{while}\,(a>0)\,\{a=a-1;\}}$$

$$\frac{\Gamma \vdash a+1 : \ell_a \sqcup L \qquad \ell_a \sqcup L \sqcup \ell \sqsubseteq \ell_a}{\Gamma, \ell \sqcup \ell_a \vdash a=a-1;}$$

$$\frac{\Gamma \vdash a-1 : \ell_a \sqcup L \qquad \ell_a \sqcup L \sqcup \ell \sqsubseteq \ell_a}{\Gamma, \ell \vdash a=a-1;}$$

Constraints:

$\ell = \ell_a \sqcup \ell_b$

$\ell_a \sqcup \bot \sqcup \ell \doteq \ell_a$

$\ell_a = \top$, constr. satisfied

Case $\ell_a = \bot$, $\ell_b = \bot$ ✓

$\ell_a = \bot$, $\ell_b = \top$ ✗

Exercise 1: Dynamic Information Flow

Consider the following function under an information flow security policy over the lattice $L \leq H$.

```
bool function f(x) {
P1  y = true;
P2  z = true;
P3  if x then y = false; P4
P5  if (!y) then z = false; P6
P7  return !z;
    }
```

We assume to consider a dynamic information flow mechanism where the typed security policy Γ is *dynamic*. We also assume that the first two assignments to variables y and z are low-level assignments (namely, the corresponding types and values are at the low level L).

- Discuss the dynamic information flow policies associated to the execution of the function f above. Consider two cases:
  1. Function f is called with the true value at the security level H;
  2. Function f is called with the false value at the security level H;

We recall the rule for assignment: $x := e \Rightarrow \Gamma(e) \sqcup pc_x \sqsubseteq \Gamma(x)$

Function entry: $\Gamma = \{x = H\}$. We assume the pc_x is low when function is called.

$P1: \{x=H, y=L\}$ $pc_x = L$

$P2: \{x=H, y=L, z=L\}$ $pc_x = L$

$P3 = P2$.

Case 1: x is true, we execute P4. pc_x is promoted to H.

- $P4: \{x=H, y=H, z=L\}$ $pc_x = H$
- $P5: \{x=H, y=H, z=L\}$ $pc_x = L$ (we restore old pc_x)
- Since y is H, pc_x is promoted again to H.
- $P6: \{x=H, y=H, z=H\}$ $pc_x = H$
- $P7: \{x=H, y=H, z=H\}$ $pc_x = L$
- Returned value is High. We return true.

Case 2: x is false, we skip. In P5, we skip again because y is false.

- $P7: \{x=H, y=L, z=L\}$ $pc_x = L$
- Returned value is Low. We return false.

Exercise 2: Static Taint Analysis

Consider the following program

```
1 x = getInput();
2 a = x;
3 b = a*a;
4 d = 10;
5 if d < a then y = 0 else y = a+1;
8 if d > b then y = y+1 else y = b-1;
  print(y);
```

(6, 7, 9, 10, 11)

- Discuss how static taint analysis is applied to discover whether the previous program fragments presents taint flow. Specifically define and illustrate the constraints generated by the static taint analysis.

Prog. Point	GEN	KILL	IN	OUT
P1	$\{x\}$	$\emptyset$	$\emptyset$	$\{x\}$
P2	$\{a\}$	$\emptyset$	$\{x\}$	$\{a,x\}$
P3	$\{b\}$	$\emptyset$	$\{a,x\}$	$\{a,x,b\}$
P4	$\emptyset$	$\{d\}$	$\{a,x,b\}$	$\emptyset$
P5	$\emptyset$	$\emptyset$	$\{a,x,b\}$	$\{a,x,b\}$
P6 $\{d<a\}=c_1$	$\emptyset$	$\{y\}$	$\{a,x,b\}$	$\{a,x,b\}$
P7 $\{!(d<a)\}=!c_1$	$\{y\}$	$\emptyset$	$\{a,x,b\}$	$\{a,x,b,y\}$

P8, case c_1:	GEN	KILL	IN	OUT
	$\emptyset$	$\emptyset$	$\{a,x,b\}$	$\{a,x,b\}$
P9, case c_2: $c_1 \wedge \{d>b\} = c_1 \wedge c_2$	$\emptyset$	$\{y\}$	$\{a,x,b\}$	$\{a,x,b\}$
P11	$\emptyset$	$\emptyset$	$\{a,x,b\}$	$\{a,x,b\}$
No violation				
P10, case $!c_2$: $c_1 \wedge (!c_2)$	$\{y\}$	$\emptyset$	$\{a,x,b\}$	$\{a,x,b,y\}$
P11	$\emptyset$	$\emptyset$	$\{a,x,b,y\}$	$\{a,x,b,y\}$
Security violation				

P8, case $!c_1$:	GEN	KILL	IN	OUT
	$\emptyset$	$\emptyset$	$\{a,x,b,y\}$	$\{a,x,b,y\}$
P9, case c_2: $(!c_1) \wedge c_2$	$\{y\}$	$\emptyset$	$\{a,x,b,y\}$	$\{a,x,b,y\}$

P11	$\emptyset$	$\emptyset$	$\{a, x, b, y\}$	$\{a, x, b, y\}$
Security violation				
P10, case $!c_2 : (!c_1) \wedge (!c_1)$	$\{y\}$	$\emptyset$	$\{a, x, b, y\}$	$\{a, x, b, y\}$
P11	$\emptyset$	$\emptyset$	$\{a, x, b, y\}$	$\{a, x, b, y\}$
Security violation				

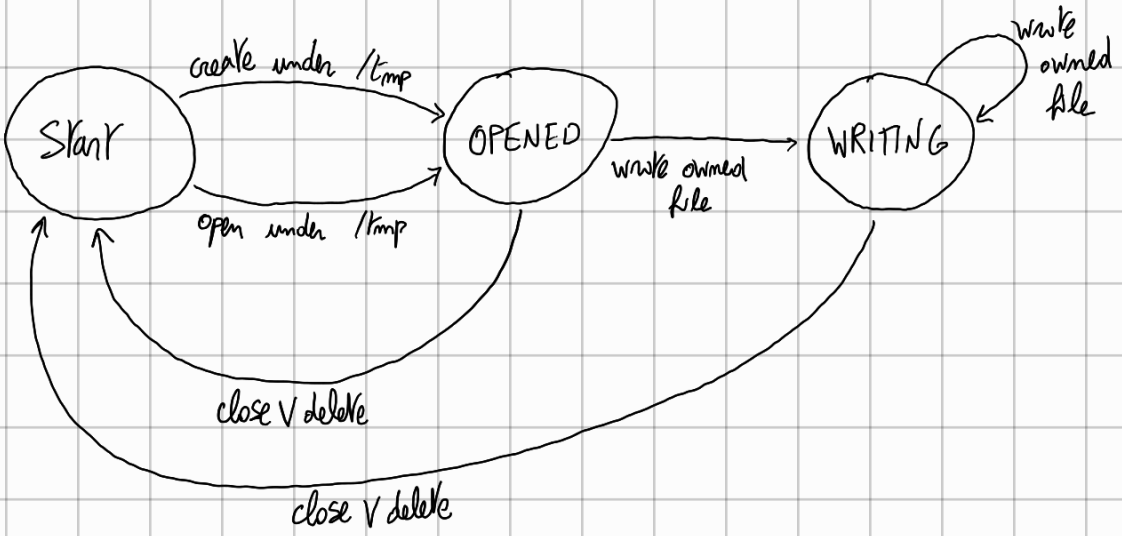
Exercise 3: Security Polices

Consider a piece of mobile code that needs to create or modify files on a local file system for its internal book-keeping operations. It is perfectly reasonable to allow untrusted mobile code to do this, as long as the files created/manipulated are unrelated to other applications. This can be ensured using a policy that enforces the following properties.

- Writes are only allowed to certain directories. The untrusted code should not be allowed to create files in arbitrary directories. The only directories allowed are those under `/tmp`
- Overwriting or deletion of a file is not permitted except for files created by the same mobile code. This ensures that untrusted mobile code does not remove files created by other applications.

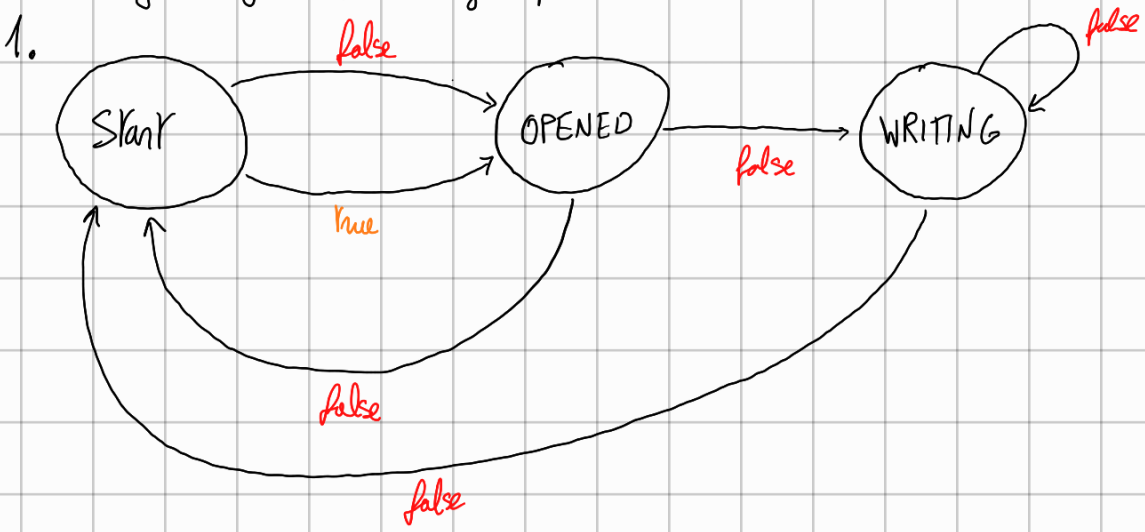
1. Design the *Security automaton* specifying the security policy discussed above.
2. Exploit the security automata defined in the previous question to instrument the code below

```
myWriter = new FileWriter(stringVal);
myWriter.write("Writing a file might be tricky");
myWriter.Close()
```



Assuming we try to open a right file (instrumentation will check)

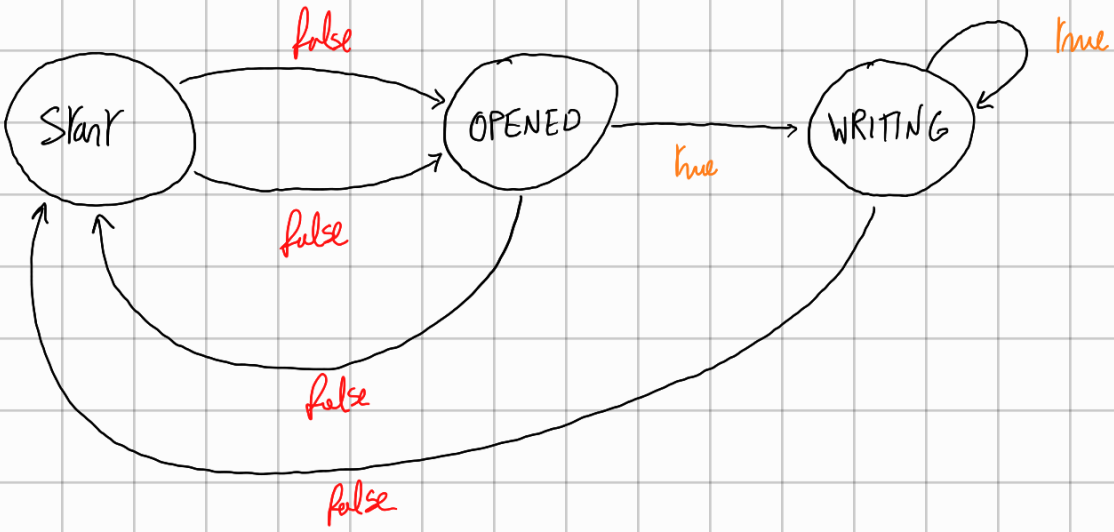
1.



"Path begins w/stringVal"

CODE: if (isPathValid(stringVal) && STATE == START) STATE = OPENED;
else abort;

2.



If we are in opened state this means we already performed a valid open.

2. if (isOwned(stringVal) && (STATE == OPENED OR STATE == WRITING))
STATE == WRITING;
else abort;

3. if (STATE == START) ABORT
else STATE = START

1. Consider the C-like code below

```
1. char buf[80];
2. int authenticated = 0;
3. // some statements
4.   void vulnerable() {
5.       gets(buf);
6. }
```

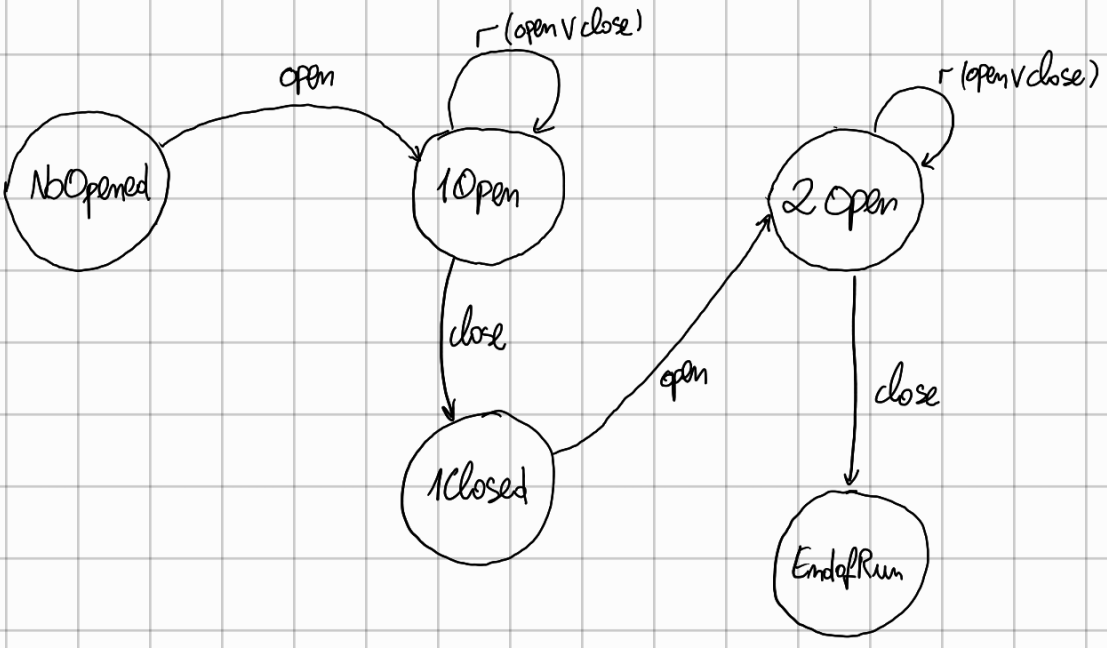
Explain the security issues related to the example above and the possible countermeasures.

The problem is the usage of the function gets, deprecated because it does no checks for the size of data that we are reading from the external world. An attacker would be free to insert as much data as possible in the buffer, even out of its bounds, resulting in buffer overflow. In this case, since we are passing a pointer to a buffer in the stack, the consequences could be buffer overflow attacks, possibly resulting in control flow hijacking, ROP attacks, stack corruption, shellcode injection or denial of service. The simplest countermeasure could be to just use a safe bounds checking alternative like fgets that requires as a parameter the max number of characters we can read. There are usually other countermeasures implemented by OS or compilers, like runtime bound checking, stack canaries, NX memory, Address Space Layout Randomization, shadow stacks.

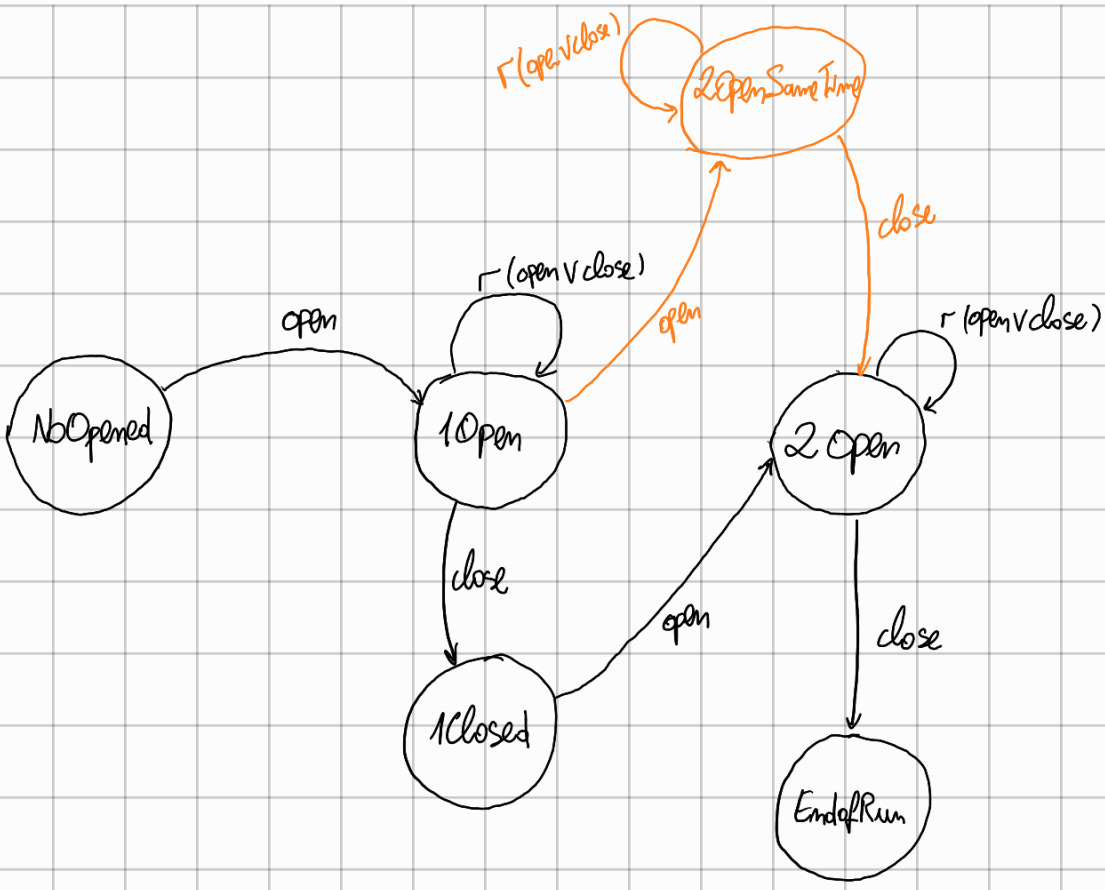
2. Assume to consider software components operating over a data repository with standard operations for opening, reading, writing and closing data objects. Define a security automata that enforces the following security policy.

   Each component must have a maximum of two data objects opened in a run.

   Illustrate the automata and discuss your design choices.



The security policy enforced by this automaton works under the assumption that you only keep one data object opened at a time. When you open the first one, you stay in 1Open as long as you don't close or open anything else. You can close your data object to go in 1Close, then you are ready to open another data object. If you do, you end up in 2Open and you can read and write. If you close the 2nd data object you can't do anything else, your run is over. If we wanted to consider the possibility of having two data objects opened at the same time, we can apply this modification:


⌐(open∨close)
2OpenSameThing
close
open
⌐(open∨close)
⌐(open∨close)
open
NbOpened
1Open
2Open
close
open
close
1Closed
EndofRun

4. Consider the following code fragment

```
1. a = read ();
2. c = read ();
3. b = ~ a & c;
4. d = a & b;
5. output d
```

Illustrate the constraint systems generated by the taint analyzer running on the code above.

5. Modify the code of example 4 as follows

```
Line 3: b = And(~ a, c)
Line 4 : d = And(a, b)
```

where And is function returning the logical and of the input parameters

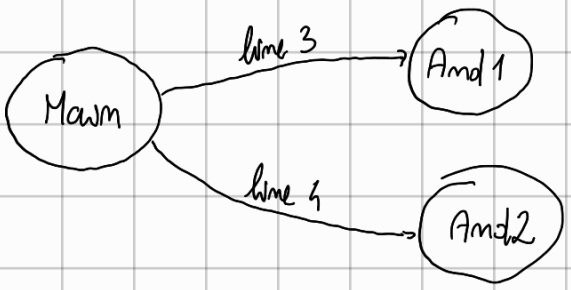
Illustrate the constraint systems generated by the taint analyzer running on the modified code.

	GEN	KILL	IN	OUT
1.	{a}	∅	∅	{a}
2.	{c}	∅	{a}	{a,c}
3.	{b}	∅	{a,c}	{a,c,b}
4.	{d}	∅	{a,c,b}	{a,c,b,d}
5.	∅	∅	{a,c,b,d}	{a,c,b,d}

$OUT(s) = (IN(s) \setminus KILL(s)) \cup GEN(s)$

Case 2: Context sensitive analysis:

CALL GRAPH:



Call sites:	Context:	Taint Out
P.3: And(a, c)	{a=tainted, c=tainted}	Yes
P.4: And(a, b)	{a=tainted, b=tainted}	Yes

The result is that in the two call sites, since we received tainted inputs we also give tainted outputs.