1 Operational semantics of WASM:

Memory operation:

$$\frac{0 \le i < |m^*_{mem}|}{\{\ \text{stack } i;\ \text{instr } i32.\text{load}\ \} \to \{\ \text{stack } m^*_{mem}[i]\ \}}$$

$$\frac{0 \le i < |m^*_{mem}|}{\{\ \text{stack } m, i;\ \text{instr } i32.\text{store}\ \} \to \{\ m^*_{mem}\ \text{with}\ [i] = m\ \}}$$

Loop:

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

br:

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

br 0:

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

br_if:

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

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

Frame $C = \{\ \text{module } m,\ \text{stack } m^*_{stack},\ \text{locals } m^*_{local},\ \text{memory } m^*_{mem},\ \text{instr } e^*_{body}\ \}$

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, $e^*_1)$, $e^*_2\}$

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

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

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

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

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

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

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

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

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

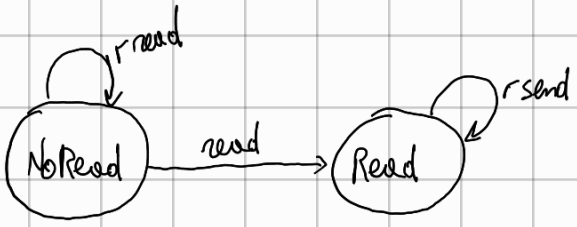
$c = \{ \text{vmstr} \quad \text{local.get } \$i \}$

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

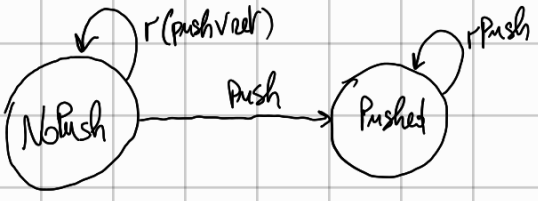
We will return value 0.

2 Inline Reference Monitors:

EX: No SENDS AFTER READ



PUSH EXACTLY ONCE BEFORE RETURNING



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

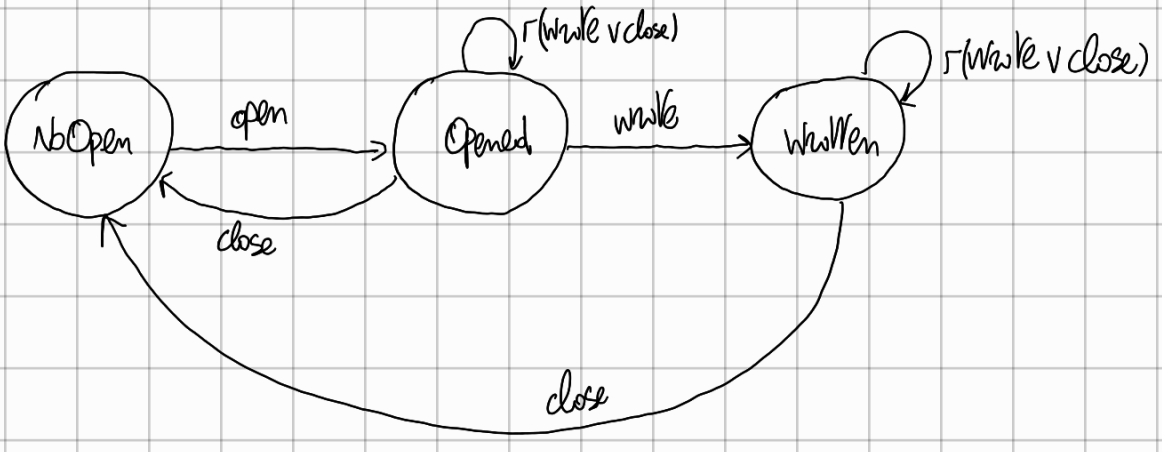
1. Design the *Security Automaton* for operating over files. A file must be opened before any operations. After it is opened, it can be written at most once, and then it can be read multiple times. It must be closed before being reopened.

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

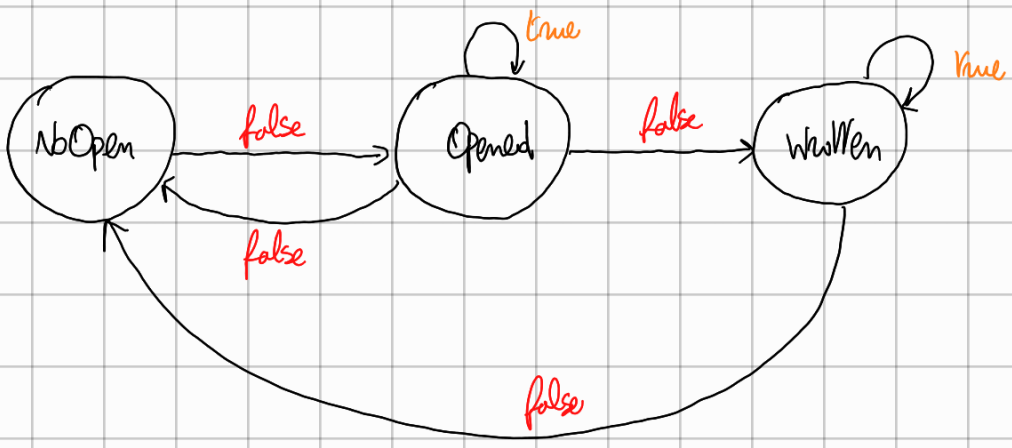
```
z=readf(f);
write(f, z1);
write(f, z2);
```

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

FSA:

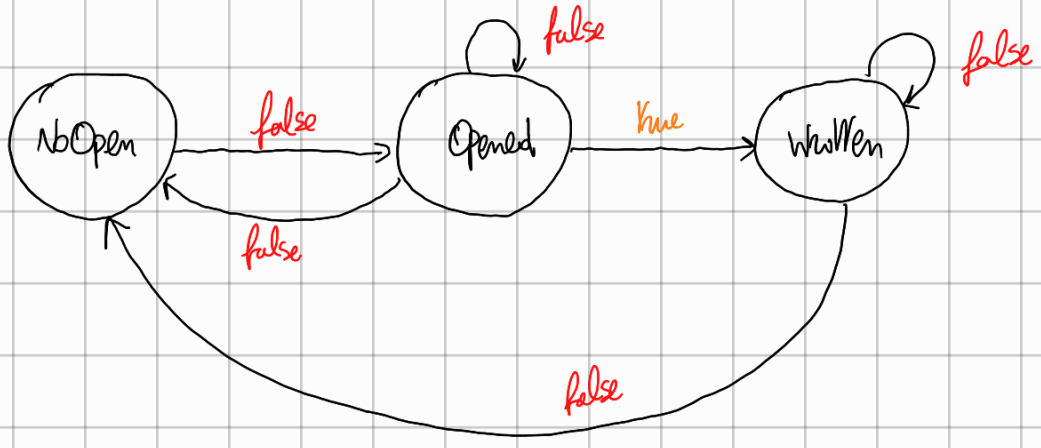


1. z = read(f);



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)

3 DTA

1. $X = 2 * \text{getInput}([20]);$
2. $y = 5 + x;$
3. goto $y;$

Point 1:

$$\frac{\gamma_M, \gamma_P, M, P \vdash 2 * \text{getInput}([20]) \Downarrow \langle 40, T\rangle \quad P' = P[x=40] \quad \gamma_P' = \gamma_P[x=T] \quad z = \Sigma[2]}{\gamma_M, \gamma_P, M, P, \Sigma, 2 : x = 2 * \text{getInput}([20]) \to \gamma_M, \gamma_P', M, P', \Sigma : 2}$$

$$\frac{src = 20 :: src'}{\gamma_M, \gamma_P, M, P \vdash \text{getInput}([20]) \Downarrow \langle 20, T\rangle}$$

Point 2:

$$\frac{\gamma_M, \gamma_P, M, P \vdash 5 + x \Downarrow \langle 45, T\rangle \quad P' = P[y=45] \quad \gamma_P' = \gamma_P[y=T] \quad \Sigma[3] = z}{\gamma_M, \gamma_P, M, P, \Sigma, 3 : y = 5 + x; \to \gamma_M, \gamma_P', M, P', \Sigma, 4 : z}$$

$$\frac{-}{\gamma_M, \gamma_P, M, P \vdash x \Downarrow \langle 40, T\rangle}$$

Point 3:

$$\frac{\gamma_M, \gamma_P, M, P \vdash y \Downarrow \langle 45, T\rangle \quad \text{IPGOTO}(T) = T \quad z = \Sigma(45)}{\gamma_M, \gamma_P, M, P, \Sigma, 4 : \text{goto } y \to \text{ERROR}}$$

not satisfied

4. Sign Analysis

1. Entry
2. var a, b, c;
3. a = 42;
4. b = 87;
5. if (input)
6. {c = a + b;}
7. else {c = a - b;}
8. Exit

$X_1 = \bot$

$X_2 = X_1[a \to T, b \to T, c \to T]$

$X_3 = X_2[a \to +]$

$X_4 = X_3[b \to +]$

$X_5 = X_4$

$X_6 = X_5[c \to eval(X_5, a+b)]$

$X_7 = X_5[c \to eval(X_5, a-b)]$

$X_8 = (X_6 \sqcup X_7)$

Iteration 1

$X_1 = \bot$

$X_2 = [a = T, b = T, c = T]$

$X_3 = [a = +]$

$X_4 = [b = +]$

$X_5 = \bot$

$X_6 = [c = \bot]$

$X_7 = [c = \bot]$

$X_8 = \bot$

Iteration 2 ... Until final

$X_1 = \bot$

$X_2 = [a = T, b = T, c = T]$

$X_3 = [a = +, b = T, c = T]$

$X_4 = [a = +, b = +, c = T]$

$X_5 = [a = +, b = +, c = T]$

$X_6 = [a = +, b = +, c = +]$

$X_7 = [a = +, b = +, c = T]$

$X_8 = [a = +, b = +, c = T]$

5. Available expressions Analysis

```
var x, y, z, a, b;
z = a+b;
y = a*b;
while y > a+b {
    a = a+1;
    x = a+b;
}
```

GEN-KILL SET ANALYSIS: $OUT(S) = (IN(S) \setminus KILL(S)) \cup GEN(S)$

	IN	OUT
1	$\emptyset$	IN(1)
2	OUT(1)	IN(2) $\cup$ {a+b}
3	OUT(2)	IN(3) $\cup$ {a*b}
4	OUT(3) $\cap$ OUT(6)	IN(4) $\cup$ {a+b}
5	OUT(4)	IN(5) $\cup$ {a+1} \ a
6	OUT(5)	IN(6) $\cup$ {a+b}

ITERATION 0:

	IN	OUT
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	$\emptyset$	$\emptyset$

ITERATION 1:

	IN	OUT
1	$\emptyset$	$\emptyset$
2	$\emptyset$	{a+b}
3	{a+b}	{a+b, a*b}
4	$\emptyset$	{a+b}
5	{a+b}	$\emptyset$
6	$\emptyset$	{a+b}

ITERATION 2:

	IN	OUT
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{a+b\}$
3	$\{a+b\}$	$\{a+b, a*b\}$
4	$\{a+b\}$	$\{a+b\}$
5	$\{a+b\}$	$\emptyset$
6	$\emptyset$	$\{a+b\}$

Monotone Framework approach

$[\text{var } x, y, z, a, b;] = [\text{entry}]$

$[z = a+b;] = ([\text{var } x, y, z, a, b;] \cup \{a+b\}) \downarrow z$

$[y = a*b;] = ([z = a+b] \cup \{a*b\}) \downarrow y$

$[\text{while } (y > a+b)] = (([y = a*b] \cap [x = a+b;]) \cup \{a+b\})$

$[a = a+1;] = ([\text{while } (y > a+b)] \cup \{a+1\}) \downarrow a$

$[x = a+b;] = ([a = a+1;] \cup \{a+b\}) \downarrow x$

Iteration 0:

$[\text{var } x, y, z, a, b;] = \emptyset$

$[z = a+b;] = \emptyset$

$[y = a*b;] = \emptyset$

$[\text{while } (y > a+b)] = \emptyset$

$[a = a+1;] = \emptyset$

$[x = a+b;] = \emptyset$

Iteration 1:

$[\text{var } x, y, z, a, b;] = \emptyset$

$[z = a+b;] = \{a+b\}$

$[y = a*b;] = \{a+b, a*b\}$

$[\text{while } (y > a+b)] = \{a+b\}$

$[a = a+1;] = \emptyset$

$[x = a+b;] = \{a+b\}$

Iteration 2:

$[\text{var } x, y, z, a, b;] = \emptyset$

$[z = a+b;] = \{a+b\}$

$[y = a*b;] = \{a+b, a*b\}$

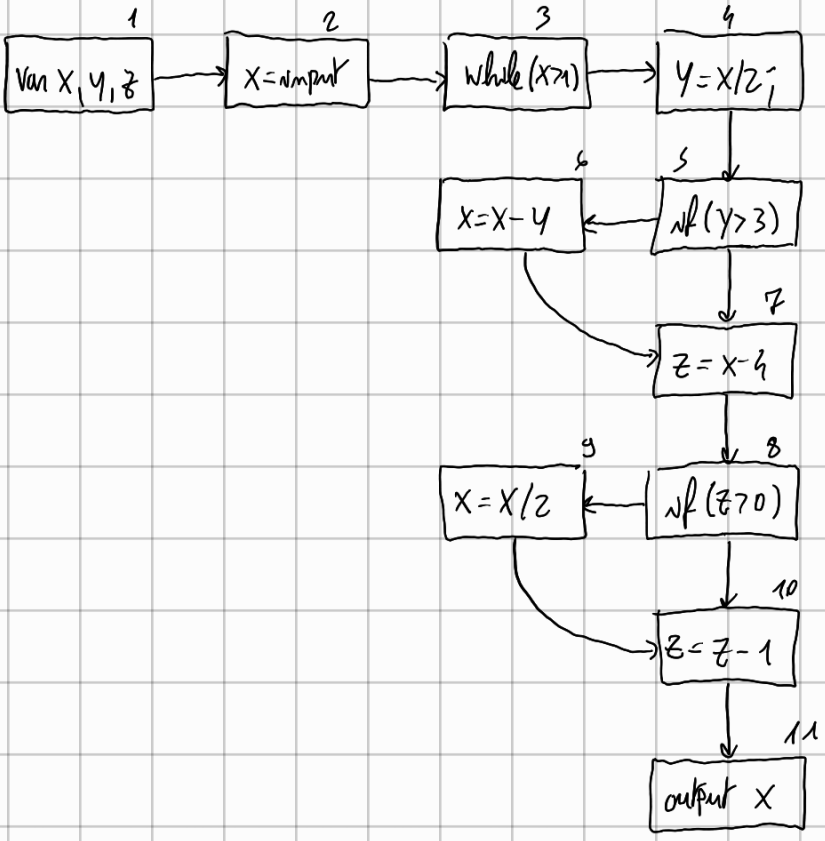
$[\text{while } (y > a+b)] = \{a+b\}$

$[a = a+1;] = \emptyset$

$[x = a+b;] = \{a+b\}$

6. Liveness Analysis

```
1. var x, y, z;
2. x = input;
3. while (x>1) {
4.   y = x/2;
5.   if (y>3) 6. x = x-y;
7.   z = x-4;
8.   if (z>0) 9. x = x/2;
10.  z = z-1;
     }
11. output x;
```



Gen-Kill method: $OUT(S) = \bigcup_{W \in Succ(S)} IN(W)$

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

	IN	OUT
1.	$OUT(1) \setminus \{x,y,z\}$	$IN(2)$
2.	$OUT(2) \setminus \{x\}$	$IN(3)$
3.	$OUT(3) \cup \{x\}$	$IN(4)$
4.	$OUT(4) \setminus \{y\} \cup \{x\}$	$IN(5)$
5.	$OUT(5) \cup \{y\}$	$IN(6) \cup IN(7)$
6.	$OUT(6) \setminus \{x\} \cup \{x,y\}$	$IN(7)$
7.	$OUT(7) \setminus \{z\} \cup \{x\}$	$IN(8)$
8.	$OUT(8) \cup \{z\}$	$IN(9) \cup IN(10)$
9.	$OUT(9) \setminus \{x\} \cup \{x\}$	$IN(10)$
10.	$OUT(10) \setminus \{z\} \cup \{z\}$	$IN(11)$
11.	$OUT(11) \cup \{x\}$	$\emptyset$

ITERATION 0:

	IN	OUT
1.	$\emptyset$	$\emptyset$
2.	$\emptyset$	$\emptyset$
3.	$\emptyset$	$\emptyset$
4.	$\emptyset$	$\emptyset$
5.	$\emptyset$	$\emptyset$
6.	$\emptyset$	$\emptyset$
7.	$\emptyset$	$\emptyset$
8.	$\emptyset$	$\emptyset$
9.	$\emptyset$	$\emptyset$
10.	$\emptyset$	$\emptyset$
11.	$\emptyset$	$\emptyset$

ITERATION 1:

	IN	OUT
1.	$\emptyset$	$\emptyset$
2.	$\emptyset$	$\{x\}$
3.	$\{x\}$	$\{x\}$
4.	$\{x\}$	$\{x,y\}$
5.	$\{x,y\}$	$\{x,y\}$
6.	$\{x,y\}$	$\{x\}$
7.	$\{x\}$	$\{x,z\}$
8.	$\{x,z\}$	$\{x,z\}$
9.	$\{x,z\}$	$\{x,z\}$
10.	$\{x,z\}$	$\{x\}$
11.	$\{x\}$	$\emptyset$

Monotone Framework:

$[\text{var } x,y,z] = \emptyset[x=\text{input}] \setminus \{x,y,z\}$

$[x=\text{input}] = ([\text{while } (x>1)]) \setminus \{x\}$

$[\text{while } (x>1)] = ([y=x/2]) \cup \{x\}$

$[y=x/2] = ([\text{if } (y>3)] \setminus \{y\}) \cup \{x\}$

$[\text{if } (y>3)] = ([x=x-y] \cup [z=x-4]) \cup \{y\}$

$[x=x-y] = ([z=x-4] \setminus \{x\}) \cup \{x,y\}$

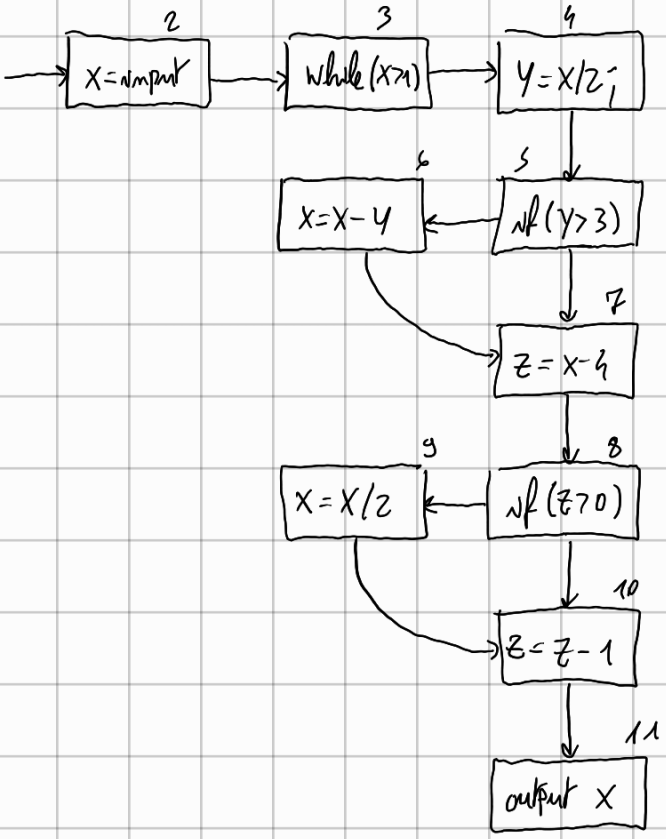
$[z=x-4] = ([\text{if } (z>0)] \setminus \{z\}) \cup \{x\}$

$[\text{if } (z>0)] = ([x=x/2] \cup [z=z-1]) \cup \{z\}$

$[x=x/2] = ([z=z-1] \setminus \{x\}) \cup \{x\}$

$[z=z-1] = ([\text{output } x] \setminus \{z\}) \cup \{z\}$

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



ITERATION 0:

$[\text{var } x,y,z] = ([x=\text{input}]) \setminus \{x,y,z\} \longrightarrow \emptyset$

$[x=\text{input}] = ([\text{while } (x>1)]) \setminus \{x\} \rightarrow \emptyset$

$[\text{while } (x>1)] = ([y=x/2]) \cup \{x\} \longrightarrow \{x\}$

$[y=x/2] = ([\text{if } (y>3)] \setminus \{y\}) \cup \{x\} \longrightarrow \{x\}$

$[\text{if } (y>3)] = ([x=x-y] \cup [z=x-4]) \cup \{y\} \rightarrow \{x,y\}$

$[x=x-y] = ([z=x-4] \setminus \{x\}) \cup \{x,y\} \longrightarrow \{x,y\}$

$[z=x-4] = ([\text{if } (z>0)] \setminus \{z\}) \cup \{x\} \longrightarrow \{x\}$

$[\text{if } (z>0)] = ([x=x/2] \cup [z=z-1]) \cup \{z\} \rightarrow \{x,z\}$

$[x=x/2] = ([z=z-1] \setminus \{x\}) \cup \{x\} \longrightarrow \{x,z\}$

$[z=z-1] = ([\text{output } x] \setminus \{z\}) \cup \{z\} \longrightarrow \{x,z\}$

$[\text{output } x] = [\text{exit}] \cup \{x\} \longrightarrow \{x\}$

7. Static Taint Analysis

1. String name = source();
2. String x;
3. if(...)
4. x = name;
5. else x = "ciao";
6. sink(x);

Case 1: Path Sensitive Analysis w/ GEN/KILL $OUT(s) = (IN(s) \setminus KILL(s)) \cup GEN(s)$

Prog. Point	GEN	KILL	IN	OUT
1.	{name}	∅	∅	{name}
2.	∅	{x}	{name}	{name}
3.	∅	∅	{name}	{name}
4. Case cond true	{x}	∅	{name}	{x, name}
5. Case cond false	∅	{x}	{name}	{name}
6. Case cond true	∅	∅	{x, name}	{x, name}
6. Case cond false	∅	∅	{name}	{name}

$AS = \{(cond, \{x, name\}), (!cond, \{name\})\}$

Taint reaches sink if cond is true.
We can make also a monotone FW analysis.

$[String\ name = source()] = [entry][name \to T] \Rightarrow \bot \Rightarrow [name = T]$
$[String\ x] = [String\ name = source()][x \to U] \Rightarrow \bot \Rightarrow [name = T, x = U]$
$[if(cond)] = [String\ x] \Rightarrow \bot \Rightarrow [name = T, x = U]$
$[x = name] = [if(cond)][x \to eval([if(cond)], name)] \Rightarrow \bot \Rightarrow [name = T, x = T]$
$[else\ x = "ciao"] = [if(cond)][x \to U] = \bot \Rightarrow [name = T, x = U]$

(cond) $[sink(x)] = [x = name] \Rightarrow \bot \Rightarrow [name = T, x = T]$

(!cond) $[sink(x)] = [else\ x = "ciao"] \Rightarrow \bot \Rightarrow [name = T, x = U]$

Example 2, with context sensitivity:

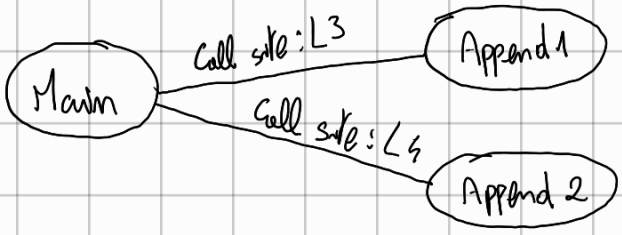
```
String a = getInput();
String b = "";
b = append(b, "abc");
b = append(b, a);
String sink = b;
println(sink);

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

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 print(b) receives a tainted input (sink). The program is not secure and cannot be executed.

(hint: reason also with path sensitivity)

(Try to perform taint analysis w/ path sensitivity with $\{(4, t), (4, f) \ldots\}$.)

Example 2:

```
String input = source();
char[] buffer = new char[32];
int i = 0;
while (i < input.length() && i < 32){
    buffer[i] = input.charAt(i);
    i = i+1;
}
String userInput = new String(buffer);
grantAccess(userInput);
```

	IN	OUT
1.	$\emptyset$	$IN(1) \cup \{input\}$
2.	$OUT(1)$	$IN(2) \setminus \{buffer\}$
3.	$OUT(2)$	$IN(3) \setminus \{i\}$
4.	$OUT(3) \cup OUT(6)$	$IN(4)$
5.	$OUT(4)$	$IN(5) \cup \{buffer\}$
6.	$OUT(5)$	$IN(6)$
7.	$OUT(4)$	$IN(7) \cup \{$ userInput if buffer tainted$\} \setminus \{$ userInput if buffer not tainted$\}$
8.	$OUT(7)$	$IN(8)$

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

$$IN(s) = \bigcup_{w \in pred(s)} OUT(w)$$

We work under the assumption that even 1 tainted character in the buffer array taints the entire buffer.

ITERATION 1

	IN	OUT
1.	$\emptyset$	{input}
2.	{input}	{input}
3.	{input}	{input}
4.	{input}	{input}
5.	{input}	{input, buffer}
6.	{input, buffer}	{input, buffer}
7.	{input}	{input}
8.	{input}	{input}

ITERATION 2

	IN	OUT
1.	$\emptyset$	{input}
2.	{input}	{input}
3.	{input}	{input}
4.	{input, buffer}	{input, buffer}
5.	{input, buffer}	{input, buffer}
6.	{input, buffer}	{input, buffer}
7.	{input, buffer}	{userInput, input, buffer}
8.	{userInput, input, buffer}	{userInput, input, buffer}

Note that if input.length is 0, we don't enter the while in a path sensitive analysis and buffer is not tainted. At point 8, OUT(s) = {input}, so tainted data does NOT reach the sink. For all the other paths (while is executed an arbitrary positive number of iterations) userInput will be tainted.

8. Type system for IF

- Prove that the provided code is well typed with respect to IF.

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

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

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

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

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

Constraints:

$\ell = \ell_a \sqcup \ell_b$

$\ell_a \sqcup L \sqcup \ell \sqsubseteq \ell_a$

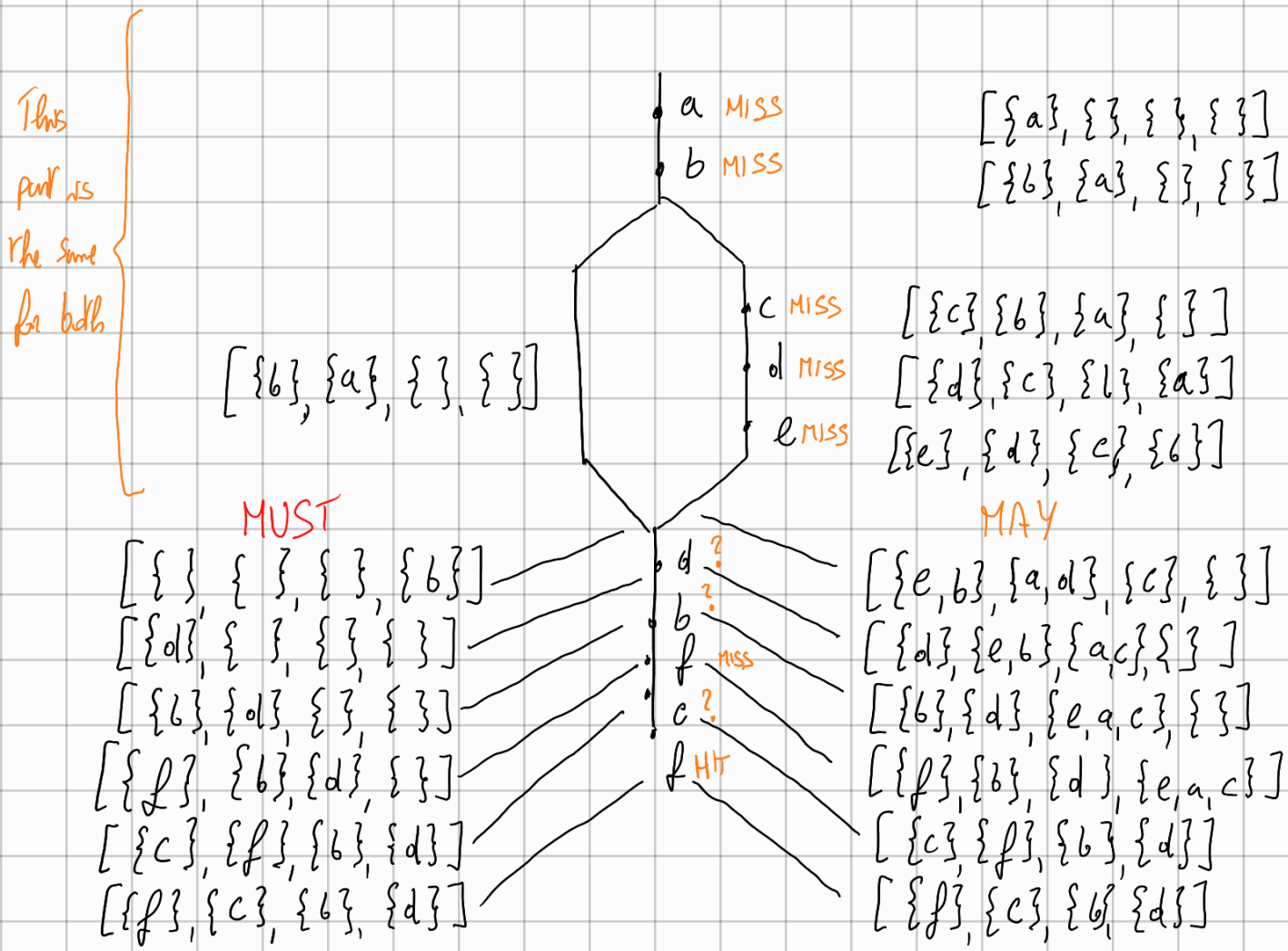
$\ell_a = H$, constr. satisfied

Case $\ell_a = L$, $\ell_b = L$ ✓

$\ell_a = L$, $\ell_b = H$ ✗

9. Cache Analysis

- Perform Must and May analysis for the CFG representation of a program



10. Dynamic information flow

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,

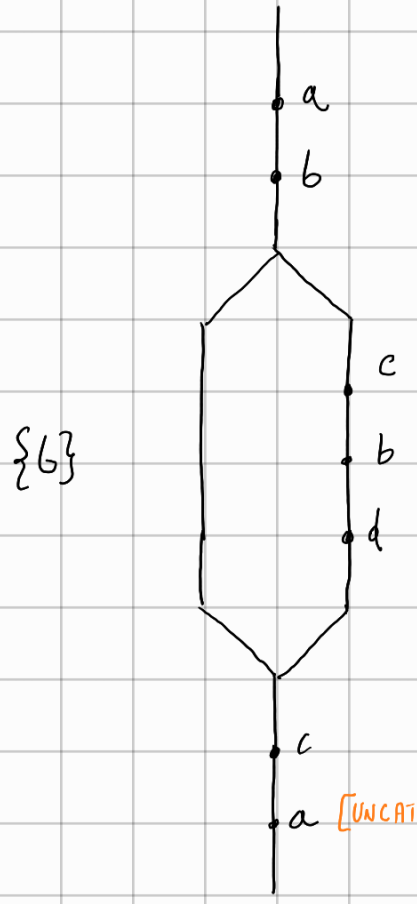
Case 2: x is false, we skip. In P5, we skip again because y is false.

P7: $\{x=H, y=L, z=L\}$ $pc_x = L$

Returned value is Low. We return false.

11. Rust related questions, mainly about borrowing and why a program wouldn't compile

[UNLIKELY] 12. Model checking for uncategorized memory accesses



ε

$\{\}$

$\{6\}$

$\{c, 6\}$

$\{6, c\}$

$\{d, b, c\}$

$\{\{6\}, \{d, 6, c\}\}$

$\{\{c, 6\}\}, \{d, b, c\}\}$

[UNCATEGORIZED ACCESS] ALWAYS HIT

13. Speculative flow

$X = a[i_1]$
$Y = a[i_2]$
$Z = X + Y$
$W = b[Z]$

Attacker's directives:

$X = load(base(a) + i_1)$
$Y = load(base(a) + i_2)$
$Z = X + Y$
$W = load(base(b) + Z)$

- Constraints:

Array access:

$$\frac{\Gamma \vdash e : S \Rightarrow K}{\Gamma \vdash a[e] : T \Rightarrow K \cup (e \subseteq S) \cup (T \subseteq a[e])}$$

Assignment:

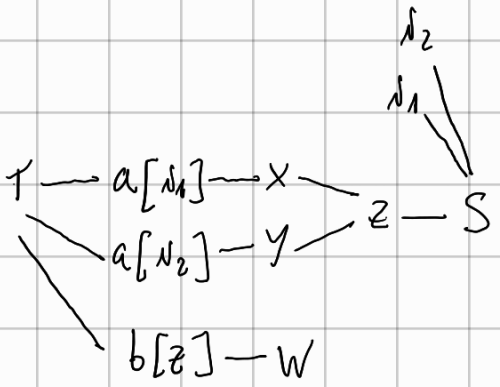
$$\frac{\Gamma \vdash r : \varphi \Rightarrow K \qquad \varphi \subseteq \Gamma(x)}{\Gamma, Pred \vdash x = r \Rightarrow K \cup (r \subseteq x)}$$

Binary Op:

$$\frac{\Gamma \vdash e_1 : \varphi_1 \Rightarrow K_1 \quad \Gamma \vdash e_2 : \varphi_2 \Rightarrow K_2 \quad \varphi_1 \subseteq \varphi \quad \varphi_2 \subseteq \varphi}{\Gamma \vdash e_1 \oplus e_2 : \varphi \Rightarrow K_1 \cup K_2 \cup (e_1 \subseteq e_1 \oplus e_2) \cup (e_2 \subseteq e_1 \oplus e_2)}$$

CONSTRAINTS:

- $N_1 \subseteq S$
- $T \subseteq a[N_1]$
- $a[N_1] \subseteq X$
- $N_2 \subseteq S$
- $T \subseteq a[N_2]$
- $a[N_2] \subseteq Y$
- $X \subseteq Z$
- $Y \subseteq Z$
- $Z \subseteq S$
- $T \subseteq b[Z]$
- $b[Z] \subseteq W$



Example 2:

```
if (X < A_sz)
  z = A[X];
if (z < B_sz)
  y = B[z];
```

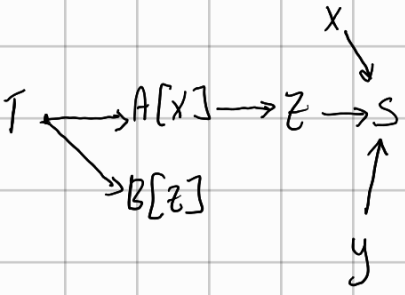
Attacker's directives:

1. guard ((X<A_sz)^true, [fwd], 1)
2. z = load (base(A) + X)
3. guard ((z < B_sz)^true, [fwd], 2)
4. y = load (base(B) + z)

The attacker will execute speculatively directive 2 and 4:

CONSTRAINTS:

- $X \subseteq S$
- $T \subseteq A[X]$
- $A[X] \subseteq Z$
- $Z \subseteq S$
- $T \subseteq B[Z]$
- $Y \subseteq S$



We will need a protect on z to cut the path from T to S.

Additional exercises:

```
(func $inf_incr (local $x i32) (param $y i32)
  ;; x = 0
  (i32.const 0)
  (set_local $x)

  (loop $incr_loop
    ;; x++
    (get_local $x)
    (i32.const 1)
    (i32.add)
    (set_local $x)
      ⋆
    (br $incr_loop)))
```

```
⋆(get_local $y)
 (get_local $x)
  i32.sub
  i32.eqz
  br_if 1
```

1. Simulate the execution of this function with actual argumnet 3.Show step-by-step stack, local variable state and the operational rule applied.

We assume that the operand 3 on top of the stack is present before the instruction call 0, assuming that module m[0] = function func.

Because of administrative ecoding, a frame is constructed. It is build like this:

$C = \{$ module m, memory m^*_{mem}, stack $\mathcal{E}$, locals $x=0, y=3$, instr i32 const 0 $\}$

We will only show what changes across executions:

$C = \{$ stack 0, instr set_local $\$x\}$
$C = \{$ stack $\mathcal{E}$, locals $x=0, y=3$, instr loop $e^*_{body}\}$
$C = \{$ instr label $\{$ loop $e^*_{body}\}$ $(\mathcal{E}$; instr get_local $\$x)\}$
$C = \{$ " $(0$; instr i32.const $1)\}$
$C = \{$ " $(1, 0$; instr i32.add$)\}$
$C = \{$ " $(1$; instr i32.set_local $\$x)\}$
$C = \{$ locals $x=1; y=3,$ " " $(\mathcal{E}$; instr get_local $\$y)\}$
$C = \{$ instr label $\{$ loop $e^*_{body}\}$ $(3$; instr get_local $\$x)\}$

$c = \{$ " $\quad$ " (1, 3 ; wasm.i32.sub) $\}$

$c = \{$ " $\quad$ " (-2 ; wasm.i32.eqz) $\}$

$c = \{$ " $\quad$ " (0 ; wasm.br_if 1) $\}$

$c = \{$ " $\quad$ " (ε ; br $ $wasm_loop) $\}$

$c = \{$ wasm.loop $\ell^{*}_{body} \}$

$c = \{$ wasm.label $\{ \text{loop } \ell^{*}_{body} \}$ (ε ; get_local $[x] \}$

⋮

We repeat until we have $y = x$;

$c = \{$ " $\quad$ " (1 ; wasm.br_if 1) $\}$

$c = \{$ stack ε, locals $x=3, y=3$, wasm.$\varepsilon \}$.

We return.

• Do Available Exp analysis

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

```
var x,y,z; } 1
x = input; }
while (x>1) {2
y = x/2; 3
if (y>3){ x = x-y;5
z = x-4; 6
if (z>0){ x = x/2;8
z = z-1;9
}
output x;10
```

	IN	OUT
1	$\emptyset$	$IN(1) \setminus exp(x)$
2	$OUT(1) \cap OUT(9)$	$IN(2) \cup \{x>1\}$
3	$OUT(2)$	$IN(3) \cup \{x/2\} \setminus exp(y)$
4	$OUT(3)$	$IN(4) \cup \{y>3\}$
5	$OUT(4)$	$IN(5) \cup \{x-y\} \setminus exp(x)$
6	$OUT(5) \cap OUT(4)$	$IN(6) \cup \{x-4\} \setminus exp(z)$
7	$OUT(6)$	$IN(7) \cup \{z>0\}$
8	$OUT(7)$	$IN(8) \cup \{x/2\} \setminus exp(x)$
9	$OUT(8) \cap OUT(7)$	$IN(9) \cup \{z-1\} \setminus \{z\}$
10	$OUT(2)$	$IN(10)$

ITERATION 1 :

	IN	OUT
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x>1\}$
3	$\{x>1\}$	$\{x>1, x/2\}$
4	$\{x>1, x/2\}$	$\{x>1, x/2, y>3\}$
5	$\{x>1, x/2, y>3\}$	$\{y>3\}$
6	$\{y>3\}$	$\{y>3, x-4\}$
7	$\{y>3, x-4\}$	$\{y>3, x-4, z>0\}$
8	$\{y>3, x-4, z>0\}$	$\{y>3, z>0\}$
9	$\{y>3, z>0\}$	$\{y>3\}$
10	$\{x>1\}$	$\{x>1\}$

ITERATION 2 :

	IN	OUT
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

- Also do sign analysis

```
var x,y,z; 1
x = input; 2
while (x>1) {3
y = x/2; 4
if (y>3) 5 x = x-y; 6
z = x-4; 8
if (z>0) 8 x = x/2; 9
z = z-1; 10
}
output x; 11
```

Assuming var x,y,z doesn't do anything to our sign analysis, otherwise we would solve it immediately.

$[\text{var } x,y,z] = [\text{entry}] = \bot$

$[x = \text{input}] = [\text{var } x,y,z][x \to \top] = \bot = [x \to \top]$

$[\text{while } (x>1)] = ([x=\text{input}] \sqcup [z=z-1]) = \bot = [x \to \top] = [x \to \top, y \to \top, z \to \top]$

$[y = x/2] = [\text{while } (x>1)][y \to \text{eval}([\text{while}(x>1)], x/2)] = \bot = [x \to \top, y \to \top] = [x \to \top, y \to \top, z \to \top]$

$[\text{if}(y>3)] = [y = x/2] = \bot = [x \to \top, y \to \top] = [x \to \top, y \to \top, z \to \top]$

$[x = x-y] = [\text{if}(y>3)][x \to \text{eval}([y=x/2], x-y)] = \bot = [x \to \top, y \to \top] = [x \to \top, y \to \top, z \to \top]$

$[z = x-4] = ([x = x-y] \sqcup [\text{if}(y>3)])[z \to \text{eval}(\text{join}\ldots), x-4] = \bot = [x \to \top, y \to \top, z \to \top]$

$[\text{if}(z>0)] = [z = x-4] = \bot = [x \to \top, y \to \top, z \to \top]$

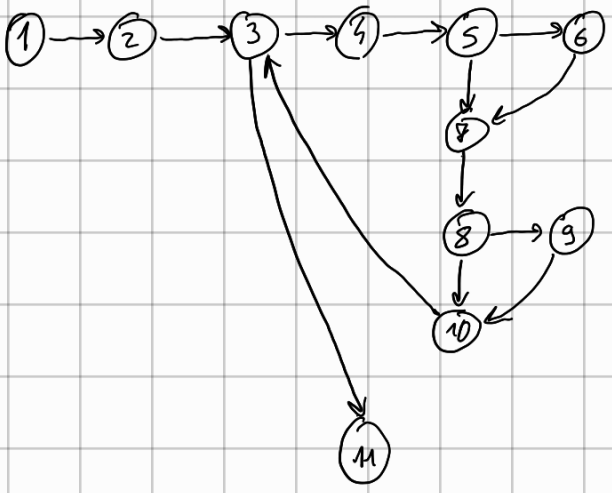
$[x = x/2] = [\text{if}(z>0)][x \to \text{eval}([\text{if}(z>0)], x/2)] = \bot = [x \to \top, y \to \top, z \to \top]$

$[z = z-1] = ([\text{if}(z>0)] \sqcup [x = x/2])[z \to \text{eval}(\text{join}(\ldots), z-1)] = \bot = [x \to \top, y \to \top, z \to \top]$

$[\text{output } x] = [\text{while } (x>1)] = \bot = [x \to \top] = [x \to \top, y \to \top, z \to \top]$

$[\text{exit}] = [\text{output } x] = \bot = [x \to \top] = [x = \top, y = \top, z = \top]$

Now liveness Analysis. Compute CFG.




```
var x,y,z; 1
x = input; 2
while (x>1) {3
y = x/2; 4
if (y>3) 5 x = x-y; 6
z = x-4; 7
if (z>0) 8 x = x/2; 9
z = z-1; 10
}
output x; 11
```

1 $[\text{var } x,y,z] = [x=\text{input}] \setminus \{x,y,z\} = \emptyset$

2 $[x=\text{input};] = [\text{while}(x>1)] \setminus \{x\} = \emptyset$

3 $[\text{while}(x>1)] = ([y=x/2] \cup [\text{output } x]) \cup \{x\} = \{x\}$

4 $[y=x/2] = [\text{if}(y>3)] \setminus \{y\} \cup \{x\} = \{x\}$

5 $[\text{if}(y>3)] = ([x=x-y;] \cup [z=x-4;]) \cup \{y\} = \{x,y\}$

6 $[x=x-y;] = [z=x-4;] \setminus \{x\} \cup \{x,y\} = \{x,y\}$

7 $[z=x-4;] = [\text{if}(z>0)] \setminus \{z\} \cup \{x\} = \{x\}$

8 $[\text{if}(z>0)] = ([x=x/2] \cup [z=z-1]) \cup \{z\} = \{x,z\}$

9 $[x=x/2;] = [z=z-1] \setminus \{x\} \cup \{x\} = \{x,z\}$

10 $[z=z-1;] = [\text{while}(x>1)] \setminus \{z\} \cup \{z\} = \{z\} = \{x,z\}$

11 $[\text{output } x;] = [\text{exit}] \cup \{x\} = \{x\}$

12 $[\text{exit}] = \emptyset$

• Make Blade analysis on Spectre V1.1

```
if(x<A_sz)
  y=A[x]
if(y<A_sz)
  z=A[y]
if(y<A_sz)
  A[y]=z;
```

Attacker's objectives:

```
guard((x<A_sz, [fwl],1)
  y=load(base(a)+x)
guard((y<A_sz, [fwl],2)
  z=load(base(a)+y)
guard((y<A_sz)^true, [fwl],3)
store(base(a)+y, z);
```

The attacker executes objectives 2, 4, 6

Rule for store:

$$\frac{\Gamma \vdash e_1 : S \Rightarrow K \qquad \Gamma \vdash e_2 : \gamma_2 \Rightarrow K_1}{\Gamma, pwr \vdash a[e_1] = e_2 : K \cup K_1 \cup (e_1 \sqsubseteq S)}$$

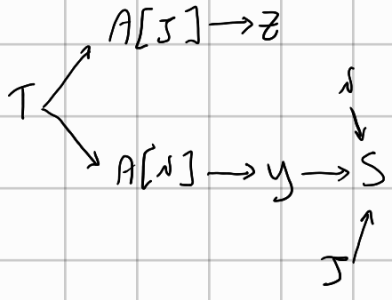
Rule for load:

$$\frac{\Gamma \vdash e : S \Rightarrow K}{\Gamma \vdash a[e] : T \Rightarrow K \cup (e \sqsubseteq S) \cup (T \sqsubseteq a[e])}$$

$$\frac{\Gamma \vdash z : \gamma \Rightarrow K \qquad \gamma \sqsubseteq \Gamma(x)}{\Gamma, pwr \vdash x = z \Rightarrow K \cup (z \sqsubseteq x)}$$

CONSTRAINTS:

- $I \sqsubseteq S$
- $T \sqsubseteq A[I]$
- $A[I] \sqsubseteq Z$
- $x \sqsubseteq S$
- $T \sqsubseteq A[x]$
- $A[x] \sqsubseteq y$
- $y \sqsubseteq S$



- Use the static type system to determine constraints for Non Interference for the variables.

while (a > b) { if (c > 0) then d = a; else if (c < -2) then b = c; else c = a; }

$$\frac{\Gamma \vdash a > b : \ell \qquad \Gamma, \ell \vdash \text{if } (c>0) \text{ then } d=a \text{ else if } (c<-2) \text{ then } b=c; \text{ else } c=a;}{\Gamma, L \vdash \text{while } (a>b) \{ \text{if } (c>0) \text{ then } d=a; \text{ else if } (c<-2) \text{ then } b=c; \text{ else } c=a; \}}$$

$$\frac{\Gamma \vdash a : l_a \quad \Gamma \vdash b : l_b}{\Gamma, L \vdash a > b : \ell = l_a \sqcup l_b} \qquad \ell = l_a \sqcup l_b$$

$$\frac{\Gamma \vdash c > 0 : l_c \qquad \Gamma, \ell \sqcup l_c \vdash d = a; \qquad \Gamma, \ell \sqcup l_c \vdash \text{if } (c<-2) \text{ then } \{b=c;\} \text{ else } \{c=a;\}}{\Gamma, \ell \vdash \text{if } (c>0) \text{ then } \{d=a;\} \text{ else } \{\text{if } (c<-2) \text{ then } \{b=c;\} \text{ else } \{c=a;\}\}}$$

$$\frac{\Gamma \vdash a : l_a \qquad l_a \sqcup l_c \sqcup \ell \sqsubseteq l_d}{\Gamma, \ell \sqcup l_c \vdash d = a;}$$

$$\frac{\Gamma \vdash c < -2 : l_c \qquad \Gamma, \ell \sqcup l_c \sqcup l_c \vdash b = c; \qquad \Gamma, \ell \sqcup l_c \sqcup l_c \vdash c = a;}{\Gamma, \ell \sqcup l_c \vdash \text{if } (c<-2) \text{ then } \{b=c;\} \text{ else } \{c=a;\}}$$

$$\frac{\Gamma \vdash c : l_c \qquad \ell \sqcup l_c \sqcup l_c \sqcup l_c \sqsubseteq l_b}{\Gamma, \ell \sqcup l_c \sqcup l_c \vdash b = c;} \qquad \frac{\Gamma \vdash a : l_a \qquad l_a \sqcup \ell \sqcup l_c \sqcup l_c \sqsubseteq l_c}{\Gamma, \ell \sqcup l_c \sqcup l_c \vdash c = a;}$$

$\ell = \ell_a \sqcup \ell_b$

$\ell \sqcup \ell_c \sqcup \ell_a \sqsubseteq \ell_d$

$\ell_c \sqcup \ell_c \sqcup \ell_c \sqcup \ell_b \sqsubseteq \ell_b$

$\ell_a \sqcup \ell_b \sqcup \ell_c \sqcup \ell_c \sqsubseteq \ell_c$

If everything is L, all good.

If ℓ_a is H, then everything else must be high.

If ℓ_b is H, everything but ℓ_a must be high.

ℓ_a	ℓ_b	ℓ_c	ℓ_d	NI
L	L	L	L	✓
L	L	L	H	✓
L	L	H	L	X
L	L	H	H	X
L	H	L	L	X
L	H	L	H	X
L	H	H	L	X
L	H	H	H	✓
H	L	L	L	X
H	L	L	H	X
H	L	H	L	X
H	L	H	H	X
H	H	L	L	X
H	H	L	H	X
H	H	H	L	X
H	H	H	H	✓

$\ell = \ell_a \sqcup \ell_b$

$\ell \sqcup \ell_c \sqcup \ell_a \sqsubseteq \ell_d$

$\ell_c \sqcup \ell_c \sqcup \ell_c \sqcup \ell_b \sqsubseteq \ell_b$

$\ell_a \sqcup \ell_b \sqcup \ell_c \sqcup \ell_c \sqsubseteq \ell_c$

- Apply DTA

1. $X = 2 * getInput([20]);$
2. $y = 5 + x;$
3. goto $y;$

$$\textcircled{1}\ \frac{M, P, \gamma_M, \gamma_P \vdash 2 * getInput([20]) \Downarrow \langle 40, T \rangle \quad P' = P[x \mapsto 40] \quad \gamma_P' = \gamma_P[x \mapsto T] \quad l = \Sigma[l]}{\Sigma, M, P, \gamma_M, \gamma_P, 2 \vdash x = 2 * getInput([20]); \rightarrow \Sigma, M, P', \gamma_M, \gamma_P', 3 \vdash l}$$

$$\frac{src = 20 :: src'}{M, P, \gamma_M, \gamma_P \vdash getInput([20]) \Downarrow \langle 20, T \rangle}$$

$$\textcircled{2}\ \frac{M, P, \gamma_M, \gamma_P \vdash 5 + x \Downarrow \langle 45, T \rangle \quad P' = P[y \mapsto 45] \quad \gamma_P' = \gamma_P[y \mapsto T] \quad l = \Sigma[3]}{\Sigma, M, P, \gamma_M, \gamma_P, 3 \vdash y = 5 + x; \rightarrow \Sigma, M, P', \gamma_M, \gamma_P', 4 \vdash l}$$

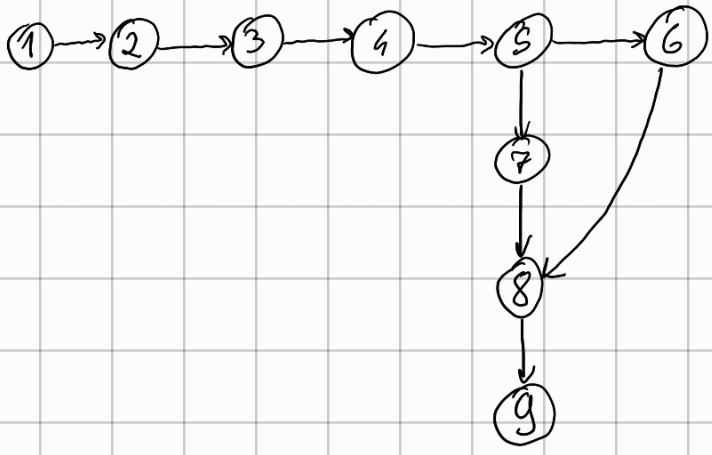
$$\frac{P(x) = 40 \quad \gamma_P(x) = T}{M, P, \gamma_M, \gamma_P \vdash x \Downarrow \langle 40, T \rangle}$$

$$\textcircled{3}\ \frac{M, P, \gamma_M, \gamma_P \vdash y \Downarrow \langle 45, T \rangle \quad TPGoto(T) = F}{\Sigma, M, P, \gamma_M, \gamma_P, 4 \vdash goto\ y \rightarrow \Sigma, M, P, \gamma_M, \gamma_P, l \rightarrow error}$$

↑ Validation

• Liveness Analysis

1. var x, y, z;
2. x = a + b;
3. y = x + 1;
4. z = x;
5. if (x > 0) {
6. z = x + 1
   } else
7. x = 2;
   }
8. y = z + x;
9. output y;



1 [var x, y, z] = [x = a + b] \ {x, y, z} = {a, b}

2 [x = a + b] = ([y = x + 1] \ {x}) ∪ {a, b} = {a, b}

3 [y = x + 1] = [z = x] \ {y} ∪ {x} = {x}

4 [z = x] = [if (x > 0)] \ {z} ∪ {x} = {x}

5 [if (x > 0)] = ([z = x + 1] ∪ [else {x = 2}]) ∪ {x} = {x, z}

6 [z = x + 1] = [y = z + x] \ {z} ∪ {x} = {x}

7 [else {x = 2}] = [y = z + x] \ {x} = {z}

8 [y = z + x] = [output y] \ {y} ∪ {z, x} = {x, z}

9 [output y] = [exit] ∪ {y} = {y}

Optimizations: y and x are never live together. We can use a single variable for it. Same for a, b and x and z, they can share registers. Plus, the assignment to y in 3 is never used later. It can be stripped.

Prova 7/ June 2024

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.

• Assume that at the beginning of execution $clk = \bot$

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

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

$$\frac{\Gamma \vdash a : \ell_a}{\Gamma \vdash a>0 : \ell_a \sqcup \bot = \ell_a}$$

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

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

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

Constraints:

$\ell = \ell_a \sqcup \ell_b$

$\ell \sqcup \ell_a \sqcup \ell_a \sqsubseteq \ell_a$

$\ell \sqcup \ell_a \sqsubseteq \ell_a$

If $\ell_a, \ell_b = L$, great.

If $\ell_a = H$, great.

If $\ell_a = L$, $\ell_b = H$ not satisfied.

Exercise 2: Static Taint Analysis

Consider the following program

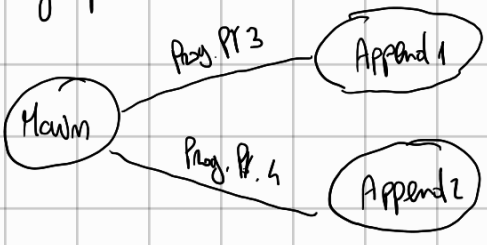
```
1 String a = getInput(); //getInput takes value from the  environment
2 String b = "";
3 b = append(b, "abc";
4 b = append(b, a)
5 String sink = b
6 println(sink);   //println is a sensible function

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

↱ SSA?

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

Call Graph:



Context Analysis:

- Append call at pr. 3: Taint status {b = unkwnked, "abc" = unkwnked};
  Taint in: No, Taint out: no.
- Append call at pr. 4: Taint status {b = unkwnked, a = kwnked};
  Taint in: Yes, Taint out: yes.

Prog. Point	GEN	KILL	IN	OUT
1	{a}	∅	∅	{a}
2	∅	{b}	{a}	{a}
3	∅	{b}	{a}	{a}
4	{b}	∅	{a}	{a,b}
5	{sink}	∅	{a,b}	{a,b,sink}
6	∅	∅	{a,b,sink}	{a,b,sink}

At point 6, s_mut receives borrowed input. That is a violation.

Consider the following Rust program

```
fn main() {
    let mut s = String::from("Hello");

    let r1 = &s;
    let r2 = &s;
    println!("r1: {}, r2: {}", r1, r2);

    let r3 = &mut s;
    println!("r1: {}, r2: {}", r1, r2);

    r3.push_str(", world!");
     println!("r3: {}", r3);
}
```

- Discuss how Rust's borrowing rules apply to the program above.

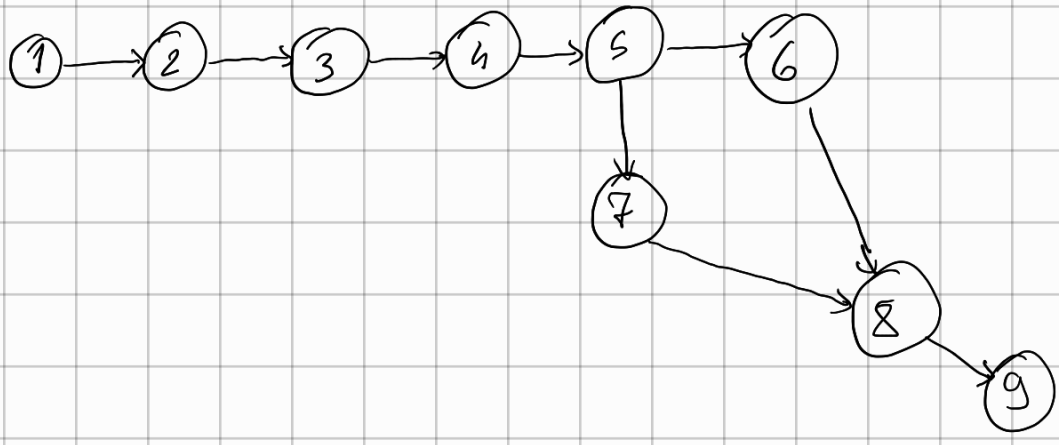
At prog. point 2, 3 we declare two immutable borrows of the variable s, which gives no problem at all. We can read from the references as we do in prog. point 4.
The code will not compile because at line 5 we try to mutably borrow the string for r3 while the immutable borrows are still valid. Even with NLL, the lifetime of the immutable borrows is over at line 6, when they are used for the last time.
To solve the compile error we just need to move line 5 after last usage of r1 and r2. This is because Rust enforces the Mutability XOR aliasing principle: you can have multiple immutable borrows valid at the same time, but only one mutable borrow can be issued, and only when other borrows are not live; this is to enforce concurrency safety.

Exercise 3: Liveness

Apply liveness analysis on the following piece of code

```
var  x,y,z; 1
x = 3 + 2; 2
y = x + 1; 3
z = x;  4
if (x > 0) {z = x + 1; } else {x = 2;}
y = z + x; 8
output  x,y,z;9
```

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



0 $[entry] = \emptyset \subseteq \emptyset$

1 $[var\ x,y,z;] = [x=3+2;] \setminus \{x,y,z\} \subseteq \emptyset \subseteq \emptyset$

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

3 $[y=x+1;] = [z=x;] \setminus \{y\} \cup \{x\} = \emptyset \subseteq \{x\}$

4 $[z=x;] \subseteq [if(x>0)] \setminus \{z\} \cup \{x\} = \emptyset \subseteq \{x\}$

5 $[if(x>0)] = ([z=x+1;] \cup [else\ \{x=2;\}]) \cup \{x\} = \emptyset \subseteq \{x\} = \{x,z\}$

6 $[z=x+1;] = [y=z+x;] \setminus \{z\} \cup \{x\} = \emptyset = \{x\}$

7 $[else\ \{x=2;\}] = [y=z+x;] \setminus \{x\} = \emptyset = \emptyset = \{z\}$

8 $[y=z+x;] = [output\ x,y,z] \setminus \{y\} \cup \{z,x\} = \emptyset = \{z,x\}$

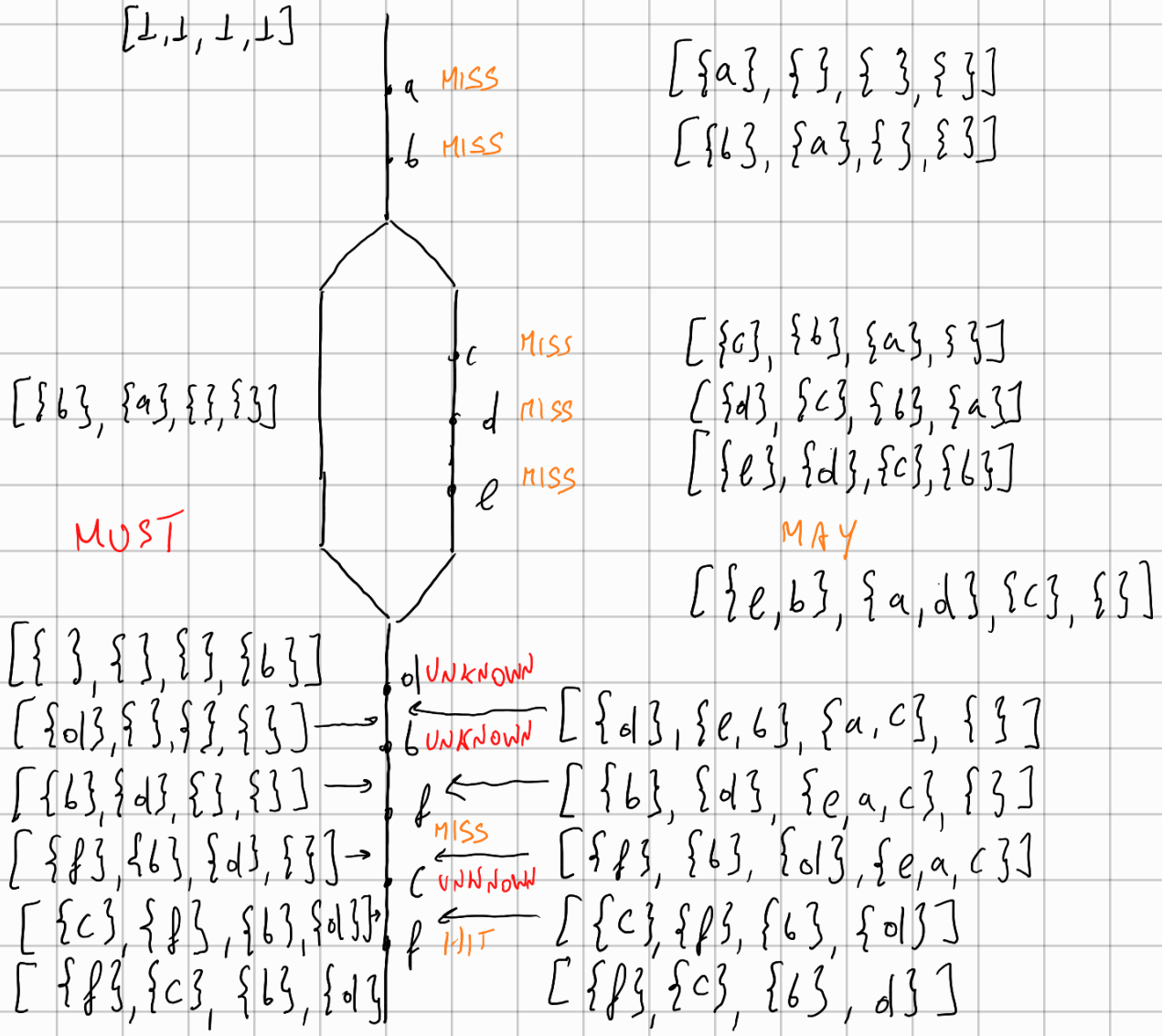
9 $[output\ x,y,z] = [exit] \cup \{x,y,z\} = \emptyset = \{x,y,z\}$

10 $[exit] = \emptyset \subseteq \emptyset$

Note: The assignment at prog. point 3 is never used. It can be deleted.

Exercise 5:

May / Must Analysis: Note, up until merge, may and must analysis coincide



Exercise 6:

```
if (x < A_sz)
    z = A[x];
if (z < B_sz)
    y = B[z];
```

Attacker's directives:

```
guard ((x < A_sz)^true, [fwl], 1);
    z = load (base(A) + x);
guard ((z < B_sz)^true, [fwl], 2);
    y = load (base(B) + z);
```

Attacker executes speculatively directives 2 and 4.

Rule for array access:

$$\frac{\Gamma \vdash e : S \Rightarrow K}{\Gamma \vdash a[e] : T \Rightarrow K \cup (e \sqsubseteq S) \cup (T \sqsubseteq a[e])}$$

Assign.

$$\frac{\Gamma \vdash e : \varphi \Rightarrow K \qquad \varphi \sqsubseteq \Gamma(x)}{\Gamma, pc \vdash x := e \Rightarrow K \cup (\tau \sqsubseteq x)}$$

CONSTRAINTS:

$x \sqsubseteq S$

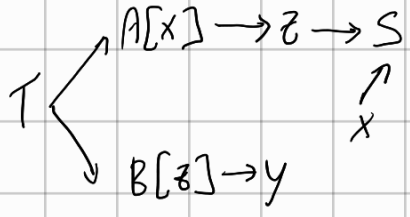
$T \sqsubseteq A[x]$

$A[x] \sqsubseteq z$

$z \sqsubseteq S$

$T \sqsubseteq B[z]$

$B[z] \sqsubseteq y$



There is a path from T to S.

To satisfy constraints, a protect should be inserted for z:

z = protect (A[x]).

Prova 19-07-2024

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) {
 y = true;
 z = true;
 if x then y = false;
 if (!y) then z = false;
 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;

Case 1:

Point 1: $\{y=L, x=H\}$

Point 2: $\{y=L, z=L, x=H\}$

Point 3: " "

Point 4: $\{y=H, z=L, x=H\}$ x is true, so ctx is high. y is updated to H.

Point 5: " ". y is false so we enter the then branch

Point 6: $\{y=H, z=H, x=H\}$ The context is H because y is H.

Point 7: " ". We return TRUE, with security level H.

Case 2:

Point 1: $\{y=L, x=H\}$

Point 2: $\{y=L, z=L, x=H\}$

Point 3: " ". X is false. We don't enter the then.

Point 5: " ". Y is true, so we don't enter the then.

Point 8: we return FALSE, with security level L.

Exercise 2: Static Taint Analysis

Consider the following program

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

- 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.

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

Prog. Points	GEN	KILL	IN	OUT
1.	$\{x\}$	$\emptyset$	$\emptyset$	$\{x\}$
2.	$\{a\}$	$\emptyset$	$\{x\}$	$\{a\}$
3.	$\{b\}$	$\emptyset$	$\{x,a\}$	$\{x,a,b\}$
4.	$\emptyset$	$\{d\}$	$\{x,a,b\}$	$\{x,a,b\}$

We now consider the 4 different paths we can follow:
let $\varphi_1 = (d < a)$ and $\varphi_2 = (d > b)$.

CASE $\varphi_1 \wedge \varphi_2$:

5.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
6.	$\emptyset$	$\{y\}$	$\{x,a,b\}$	$\{x,a,b\}$
8.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
9.	$\emptyset$	$\{y\}$	$\{x,a,b\}$	$\{x,a,b\}$
11.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$

No Taint reaches sink.

CASE $(!\varphi_1) \wedge \varphi_2$:

5.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
7.	$\{y\}$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b,y\}$
8.	$\emptyset$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$
9.	$\{y\}$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$
11.	$\emptyset$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$

Taint reaches sink.

CASE $\psi_1 \wedge (!\psi_2)$

5.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
6.	$\emptyset$	$\{y\}$	$\{x,a,b\}$	$\{x,a,b\}$
8.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
10.	$\{y\}$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b,y\}$
11.	$\emptyset$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$

Taint reaches sink

CASE $(!\psi_1) \wedge (!\psi_2)$

5.	$\emptyset$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b\}$
7.	$\{y\}$	$\emptyset$	$\{x,a,b\}$	$\{x,a,b,y\}$
8.	$\emptyset$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$
10.	$\{y\}$	$\emptyset$	$\{x,a,b,y\}$	$\{x,a,b,y\}$

Taint reaches sink

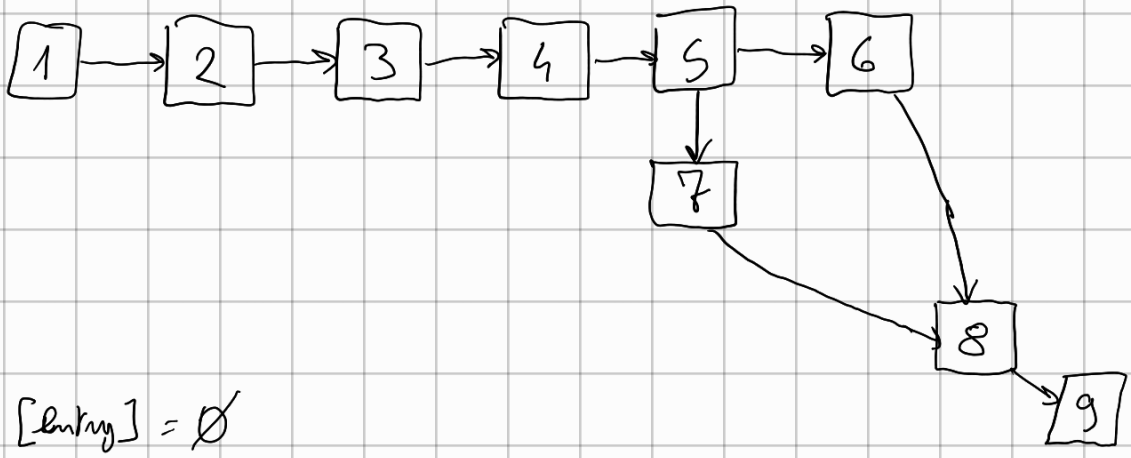
Exercise 4: Liveness Analysis

Perform liveness analysis on the following piece of code

```
1: var  x,y,z;
2: x = a + b
3: y = x + 1
4: z = x
5: if (x > 0) {
6: z = x + 1
} else {
7: x = 2
}
8: y = z + x
9: output  y;
```

- drawing its control flow graph,
- computing the constraints for each block, and
- computing the least solution of the constraints.

[*Optional*] Do the results of liveness analysis on this code enable any optimization opportunities? If so, describe the optimization.



0 [entry] = ∅

1 [var x, y, z;] = [x = a+b] \ {x, y, z} = {a, b}

2 [x = a+b] = [y = x+1] \ {x} ∪ {a, b} = {a, b}

3 [y = x+1] = [z = x] \ {y} ∪ {x} = {x}

4 [z = x] = [if (x > 0)] \ {z} ∪ {x} = {x}

5 [if (x > 0)] = ([z = x+1] ∪ [else {x = 2}]) ∪ {x} = {x, z}

6 [z = x+1] = [y = z+x] \ {z} ∪ {x} = {x}

7 [else {x = 2}] = [y = z+x] \ {x} = {z}

8 [y = z+x] = [output y] \ {y} ∪ {x, z} = {x, z}

9 [output y] = [exit] ∪ {y} = {y}

10 [exit] = ∅

y and x/z are never live at the same time: they can share the same register. Same for a, b and x, z.
Plus, assignment at 3 is not useful, y is dead until point 9.
So you can remove it.

Exercise 1: Flow Types

Consider the following program under the typing context $\Gamma = \{a = H, b = L, c = \ell\}$ where ℓ is a variable ranging over the lattice $L \leq H$.

```
if (a < b) then {
                if (b < a) then c = 0 else c = 1;
                } else {
                  if (a < b) the c = 0 else c = 1; }
```

- For which value of the variable ℓ is the above program well-typed by considering the rules of the static type system presented during the lectures? Discuss the constraints obtained from the type derivation.
- Although the static type system presented in class may reject some programs that satisfy noninterference, it is sound: it will never accept a program that violates the policy. Discuss an example where tracking information flow dynamically at runtime might mitigate some of the "false" rejections.

$\ell \sqcup L = \ell$

$$\frac{\Gamma \vdash (a<b) : \ell \quad \Gamma, \ell \vdash \text{if}(b<a) \text{ then } c=0 \text{ else } c=1; \quad \Gamma, \ell \vdash \text{if}(a<b) \text{ then } c=0 \text{ else } c=1;}{\Gamma, L \vdash \text{if}(a<b) \text{ then } \{\text{if}(b<a) \text{ then } c=0 \text{ else } c=1;\} \text{ else } \{\text{if}(a<b) \text{ then} \ldots\}}$$

$$\frac{\Gamma \vdash a : H \quad \Gamma \vdash b : L}{\Gamma \vdash a<b : L \sqcup H = H}$$

Same reasoning. $H \sqcup H = H$

$$\frac{\Gamma \vdash (b<a) : H \quad \Gamma, H \vdash c=0 \quad \Gamma, H \vdash c=1;}{\Gamma, H \vdash \text{if}(b<a) \text{ then } c=0 \text{ else } c=1;}$$

$$\frac{\Gamma \vdash 0 : L \quad H \sqcup L \sqsubseteq \ell_c}{\Gamma, H \vdash c=0}$$

Same reasoning for $c=1;$

as before

$$\frac{\Gamma \vdash (a<b) : H \quad \Gamma, H \vdash c=0 \quad \Gamma, H \vdash c=1;}{\Gamma, H \vdash \text{if}(a<b) \text{ then } c=0 \text{ else } c=1;}$$

Same rules as before

Constraints: $H \sqsubseteq \ell_c$. c cannot be low.

Exercise 2: Static Taint Analysis

Consider the following program

```
void printVal(untainted int} {...};

1 if (a == 5) then {
                  if  (b == 1) then {z = 8;} else {z = c;}
                  } else  {z = 7;}
    printVal(z);
```

- Assume that variables a, b, c are bound to tainted values. Discuss how static taint analysis is applied to discover whether the previous program fragments exhibit taint flow. Specifically define and illustrate the constraints generated by the static taint analysis.

We assume to start with $a, b, c = T$.

$IN(1) = \{a, b, c\}$. Let $(a == 5) = \varphi_1$, and $(b == 1) = \varphi_2$

Path 1: $\varphi_1 \wedge \varphi_2$

Prog. Points	GEN	KILL	IN	OUT
1	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$
2	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$
3	$\emptyset$	$\{z\}$	$\{a,b,c\}$	$\{a,b,c\}$
6	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$.

No taint reaches sinks.

Path 2: $\varphi_1 \wedge (!\varphi_2)$

1	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$
2	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$
4	$\{z\}$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c,z\}$
6	$\emptyset$	$\emptyset$	$\{a,b,c,z\}$	$\{a,b,c,z\}$

Taint reaches sink.

Path 3: $!\varphi_1$

1	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$
5	$\emptyset$	$\{z\}$	$\{a,b,c\}$	$\{a,b,c\}$
6	$\emptyset$	$\emptyset$	$\{a,b,c\}$	$\{a,b,c\}$

No fixed reaches satisfy.

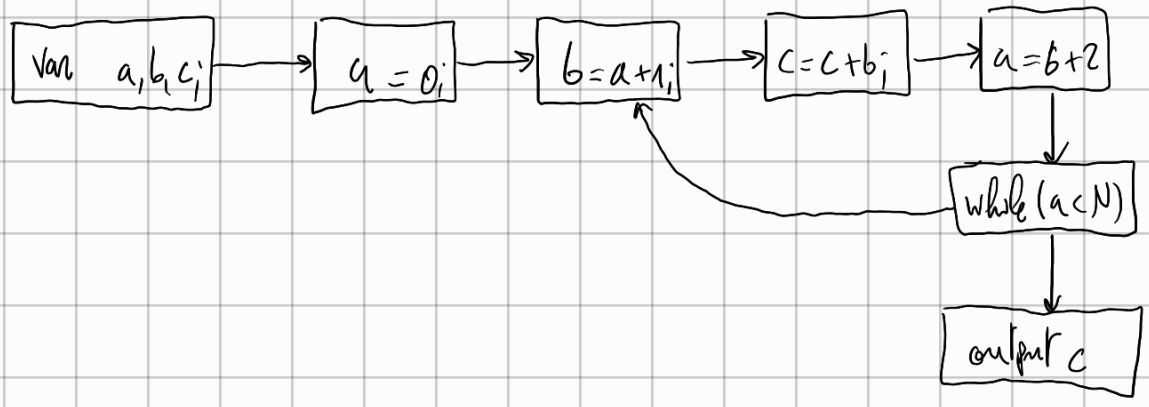
Exercise 4: Liveness analysis

Perform liveness analysis on the following piece of code

```
var  a,b,c; 1
a = 0; 2
do  {
b = a + 1;3
c = c + b;4
a = b + 2; 5
}  while (a < N);6
output  c;7
```

- drawing its control flow graph,
- computing the constraints for each block, and
- computing the least solution of the constraints.

[Optional] Do the results of liveness analysis on this code enable any optimization opportunities? If so, describe the optimization.



$[entry] = \emptyset$

$[var\ a,b,c;] = [a=0;] \setminus \{a,b,c\} = \emptyset$

$[a=0;] = [b=a+1;] \setminus \{a\} = \emptyset = \{c\}$

$[b=a+1;] = [c=c+b;] \setminus \{b\} \cup \{a\} = \emptyset = \{c,a\}$

$[c=c+b;] = [a=b+2;] \setminus \{c\} \cup \{b,c\} = \emptyset = \{c,b\}$

$[a=b+2;] = [while\ (a<N)] \setminus \{a\} \cup \{b\} = \emptyset = \{c,b\}$

$[while\ (a<N)] = ([b=a+1;] \cup [output\ c]) \cup \{a\} = \emptyset = \{c,a\}$

$[output\ c] = [exit] \cup \{c\} = \emptyset = \{c\}$

$[exit] = \emptyset$

Optimization: a, b are never live together. They can share the same variable

Exercise:

```
s = 0;
while s < 10 do
    if s = 5 then s = s + 1      } C2
             else skip;          }
    s = s + 1;                   } C3;
```

$$\frac{\Gamma, L \vdash s = 0; \qquad \Gamma, L \vdash C_2;}{\Gamma, L \vdash s = 0; C_2;}$$

$$\frac{\Gamma \vdash 0 : L \quad L \sqsubseteq \Gamma(s) = L}{\Gamma, L \vdash s = 0;} \checkmark$$

$$\frac{\Gamma, L \vdash \text{while}(s<10)\ \text{do}\ \{\text{if}(s==5)\ \text{then}\ s=s+1;\ \text{else skip}\} \qquad \Gamma, L \vdash C_3;}{\Gamma, L \vdash \text{while}\ (s<10)\ \text{do}\ \{\text{if}\ (s==5)\ \text{then}\ s=s+1;\ \text{else skip};\ C_3;\}}$$

$$\frac{\Gamma \vdash s+1 : L \quad L \sqcup L \sqsubseteq \Gamma(s) = L}{\Gamma, L \vdash s = s + 1;} \checkmark \leftarrow C_3$$

$$\frac{\Gamma \vdash (s<10) : \Gamma(s) = L \qquad \Gamma, L \vdash \{\text{if}\ (s==5)\ \text{then}\ s=s+1;\ \text{else skip}\};\ C_3;}{\Gamma, L \vdash \text{while}\ (s<10)\ \text{do}\ \{\text{if}\ (s==5)\ \text{then}\ s=s+1;\ \text{else skip}\};C_3;}$$

$$\frac{\Gamma, L \vdash (s==5) : \ell_s \sqcup \ell_5 = H \qquad \Gamma, L \sqcup H \vdash s = s+1; \qquad \Gamma, L \sqcup H \vdash \text{skip};}{\Gamma, L \vdash \text{if}\ (s==5)\ \text{then}\ s = s+1;\ \text{else skip};}$$

$$\frac{-}{\Gamma, H \vdash \text{skip}} \checkmark \qquad \frac{\Gamma \vdash s : L \quad H \sqsubseteq \Gamma(s) = L}{\Gamma, H \vdash s = s+1;} \times \text{ Not well typed}$$

Exercise 2:

H L ? L

if(a==b) {d=c+1;} else { d=c+a;};
b=c;

$$\frac{\Gamma \vdash (a==b) : \ell \qquad \Gamma, \ell \vdash d=c+1; \qquad \Gamma, \ell \vdash d=c+a;}{\Gamma, L \vdash \text{if}(a==b)\{d=c+1;\}\text{else}\{d=c+a;\}}$$

$$\frac{\Gamma \vdash a : \ell_a \qquad \Gamma \vdash b : \ell_b}{\Gamma \vdash (a==b) : \ell = \ell_a \sqcup \ell_b} \qquad \frac{\Gamma \vdash c+1 : \ell_c \qquad \ell_c \sqcup \ell \sqsubseteq \ell_d}{\Gamma, \ell \vdash d=c+1;}$$

$$\frac{\Gamma \vdash (c+a) : \ell_a \sqcup \ell_c \qquad \ell_a \sqcup \ell_c \sqcup \ell \sqsubseteq \ell_d}{\Gamma, \ell \vdash d=c+a;} \qquad \frac{\Gamma \vdash c : \ell_c \qquad L \sqcup \ell_c \sqsubseteq \ell_b}{\Gamma, L \vdash b=c;}$$

$\ell_c \sqsubseteq \ell_b$

$\ell_a \sqcup \ell_b \sqcup \ell_c \sqsubseteq \ell_d$

$\ell_a \sqcup \ell_b \sqcup \ell_c \sqcup \ell_a \sqsubseteq \ell_d$

Se null→L, ✓. Se null→H, ✓.

Se $\ell_b = H$, $\ell_c = L$, nope.

l_a	l_b	l_c	l_d	
L	L	L	L	✓
L	L	L	H	✓
L	L	H	L	✗
L	L	H	H	✗
L	H	L	L	✗
L	H	L	H	✓
L	H	H	L	✗
L	H	H	H	✓
H	L	L	L	✗
H	L	L	H	✓
H	L	H	L	✗
H	L	H	H	✗
H	H	L	L	✗
H	H	L	H	✓
H	H	H	L	✗
H	H	H	H	✓

$l_c \subseteq l_b$

$l_a \sqcup l_b \sqcup l_c \subseteq l_d$