Faculty of Mathematics and Computer Science, University of Bucharest
Alexandru Mocanu

Special Topics in Logics and Security
Seminar Exercises

Exercise 1
Give examples of partial order sets (posets).

Solution

$(\mathbb{R}, \leq)$; $(\mathbb{N}, |)$; $(\mathcal{P}(X), \subseteq)$; $(\mathbb{R} \times \mathbb{R}, \leq)$

Exercise 2
Check that the following tuplet ($\mathscr{P}$(X), ⊆) defines a poset for any given set X.

Solution

A partial order set is a relationship that is: reflexive, antisymmetric, transitive.

Reflexive: A⊆A for any A∈$\mathscr{P}$(X) ⇒ TRUE

Antisymmetric: A⊆B and B⊆A for any A,B ∈$\mathscr{P}$(X) ⇒ TRUE

Transitive: A⊆B and B⊆C for any A,B,C ∈$\mathscr{P}$(X) ⇒ TRUE

Exercise 3
Prove that the tuple (S,R) is not a poset, where S is the set of strings over an alphabet Σ (with the length of the alphabet greater or equal to 2) and R is defined by the following rule: sRt ⇔ l(s) ≤ l(t).

Solution

Consider Σ={a,b,...}, to prove that the given tuple is not a poset, we have to come with a counter-example. Let's consider s_1="ab" and s_2="ba". It is true that s_1Rs_2 and s_2Rs_1, but obviously $s_1 \neq s_2$, so the given relationship is not antisymmetric, therefore the given tuple is not a poset.

Exercise 4

Give examples of lattices.

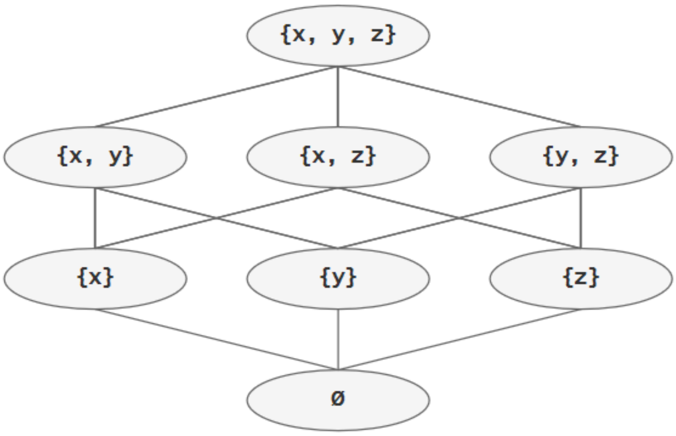
Solution

$(\mathbb{R}, max, min)$; $(\mathbb{N}, gcd, lcm)$; $(\mathcal{P}(X), \cup, \cap)$; $(\mathbb{R} \times \mathbb{R}, max, min)$

Exercise 5

Draw the lattice of a superset for a set of three elements.

Solution

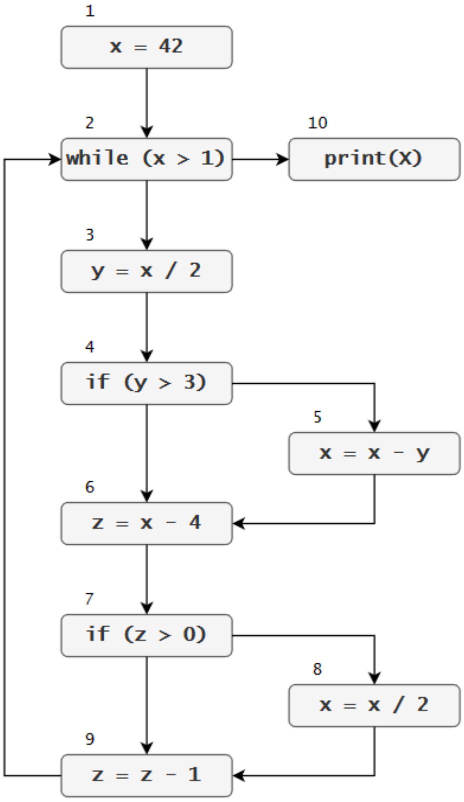


Exercise 6

(Course exercise) Apply one algorithm for live analysis on the following program:

```
 1.    x = 42;
 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.    print(x);
```

Solution



First we should make the CFG (Control Flow Graph) that models the program behaviour.

Now, we will use the formula from the course to express the live variable from each node:

$$v_n = \left(\bigcup \{ v_i \mid i \text{ succesor al lui } n \} \setminus Written(n) \right) \cup Read(n)$$

Remember that v_n are the variables that are live (i.e. their value will be used before it would be overwritten) in the state n. One should also notice that all v_n are subsets of {x,y,z} (the set of variables from the program).

$$\begin{aligned}
v_1 &= v_2 \setminus \{x\} \\
v_2 &= v_3 \cup v_{10} \cup \{x\} \\
v_3 &= v_4 \setminus \{y\} \cup \{x\} \\
v_4 &= v_5 \cup v_6 \cup \{y\} \\
v_5 &= v_6 \setminus \{x\} \cup \{x, y\} \\
v_6 &= v_7 \setminus \{z\} \cup \{x\} \\
v_7 &= v_8 \cup v_9 \cup \{z\} \\
v_8 &= v_9 \setminus \{x\} \cup \{x\} \\
v_9 &= v_{10} \setminus \{z\} \cup \{z\} \\
v_{10} &= \{x\}
\end{aligned}$$

The formulas from the table give us the definition for a function that is monotone in all of its n arguments. On this function, we will apply the least fixed point theorem for lattices using the naive algorithm.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$\perp$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$f(\perp)$	$\emptyset$	$\{x\}$	$\{x\}$	$\{y\}$	$\{x,y\}$	$\{x\}$	$\{z\}$	$\{x\}$	$\{z\}$	$\{x\}$
$f^2(\perp)$	$\emptyset$	$\{x\}$	$\{x\}$	$\{x,y\}$	$\{x,y\}$	$\{x\}$	$\{x,z\}$	$\{x,z\}$	$\{x,z\}$	$\{x\}$
$f^3(\perp)$	$\emptyset$	$\{x\}$	$\{x\}$	$\{x,y\}$	$\{x,y\}$	$\{x\}$	$\{x,z\}$	$\{x,z\}$	$\{x,z\}$	$\{x\}$

When we encounter two rows that are equal, we stop the iterative computation. The last row will give us the least fixed point, which tells us what variables are live in each state.

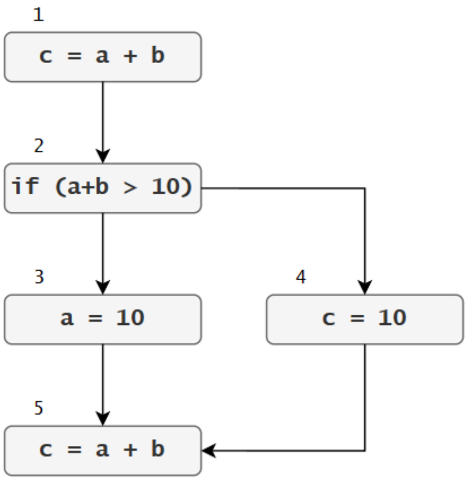
Exercise 7

Perform an available expression analysis on the following program:

```
        int a, b, c;
1.      c = a+b;
2.      if (a+b > 10)
3.              a = 10;
        else
4.              c = 10;
5.      c = a+b;
```

Solution

Available expression analysis is strongly related with live variable analysis. The goal is to find for a given state, which expressions are already computed in order to optimise the computations. A simple example can be seen in the example from above, where the expression a+b is computed and assigned to c (line 2) and afterwards the expression is computed again in the if statement (line 3).



Let's first model the program as a CFG.

Next, we need to find a formula to express the available expression (similar to the one from the course for live variables). Intuitively, the formula states that an expression is available in a given state, if it was computed previously. We exclude an expression from the available set when one of the variables from it changes its value.

$v_n = Join(v_n) \cup SubExpr(n) \setminus ChangedExpr(n)$
$Join(v_n) = \cap\{v_i | i \text{ predecessor of } n\}$
$SubExpr(n)$ = all (sub)expressions that appear in state n (i.e. x+y, y-z, x+y-z, for instruction v=x+y-z)
$ChangedExpr(n)$ = all expressions that contain a variable that is written in state n (i.e. x+y, for instruction x=0)

Note that in this case, the lattice that we are working on is P(Expr), where Expr is the set of all expressions available in the program. In the current case, we have Expr={a+b}. Now, we are going to explicit the formulas for each state and apply the least fixed point theorem to reach our solution.

$$\begin{aligned}
&v_1 = \{a+b\}\\
&v_2 = v_1 \cup \{a+b\}\\
&v_3 = v_2 \setminus \{a+b\}\\
&v_4 = v_2\\
&v_5 = v_3 \cap v_4 \cup \{a+b\}
\end{aligned}$$

	v_1	v_2	v_3	v_4	v_5
$\perp$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$f(\perp)$	$\{a+b\}$	$\{a+b\}$	$\emptyset$	$\emptyset$	$\{a+b\}$
$f^2(\perp)$	$\{a+b\}$	$\{a+b\}$	$\emptyset$	$\{a+b\}$	$\{a+b\}$
$f^3(\perp)$	$\{a+b\}$	$\{a+b\}$	$\emptyset$	$\{a+b\}$	$\{a+b\}$

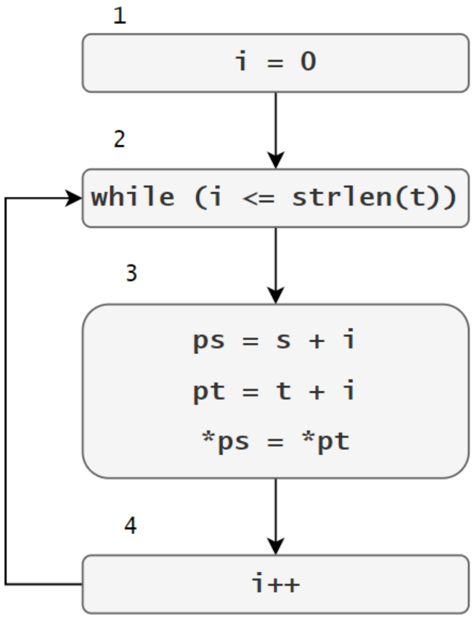
Similar to the previous exercise, we stop computing when two consecutive rows are equal. In this case the last row gives us the available expressions from each state.

Exercise 8

Analyse the following implementation of the `strcpy` function using value ranges. Decide if the program has a buffer overflow vulnerability or not and justify your answer.

```
function strcpy(char* s, char* t) {
      char* ps, pt; int i;
      for(i=0; i<=strlen(t); i++) {
            ps = s + i;
            pt = t + i;
            *ps = *pt;
      }
}
```

Solution



Let’s first model the program as a CFG.

To simplify the computations, we are going to merge all instructions from the for loop into a single block. You can try to solve this exercise without this step.

We are going to make the following notations:

s = $s_0s_1s_2s_3\ldots s_n$ (strlen(s) = n)

t = $t_0t_1t_2\ldots t_m$ (strlen(t) = m)

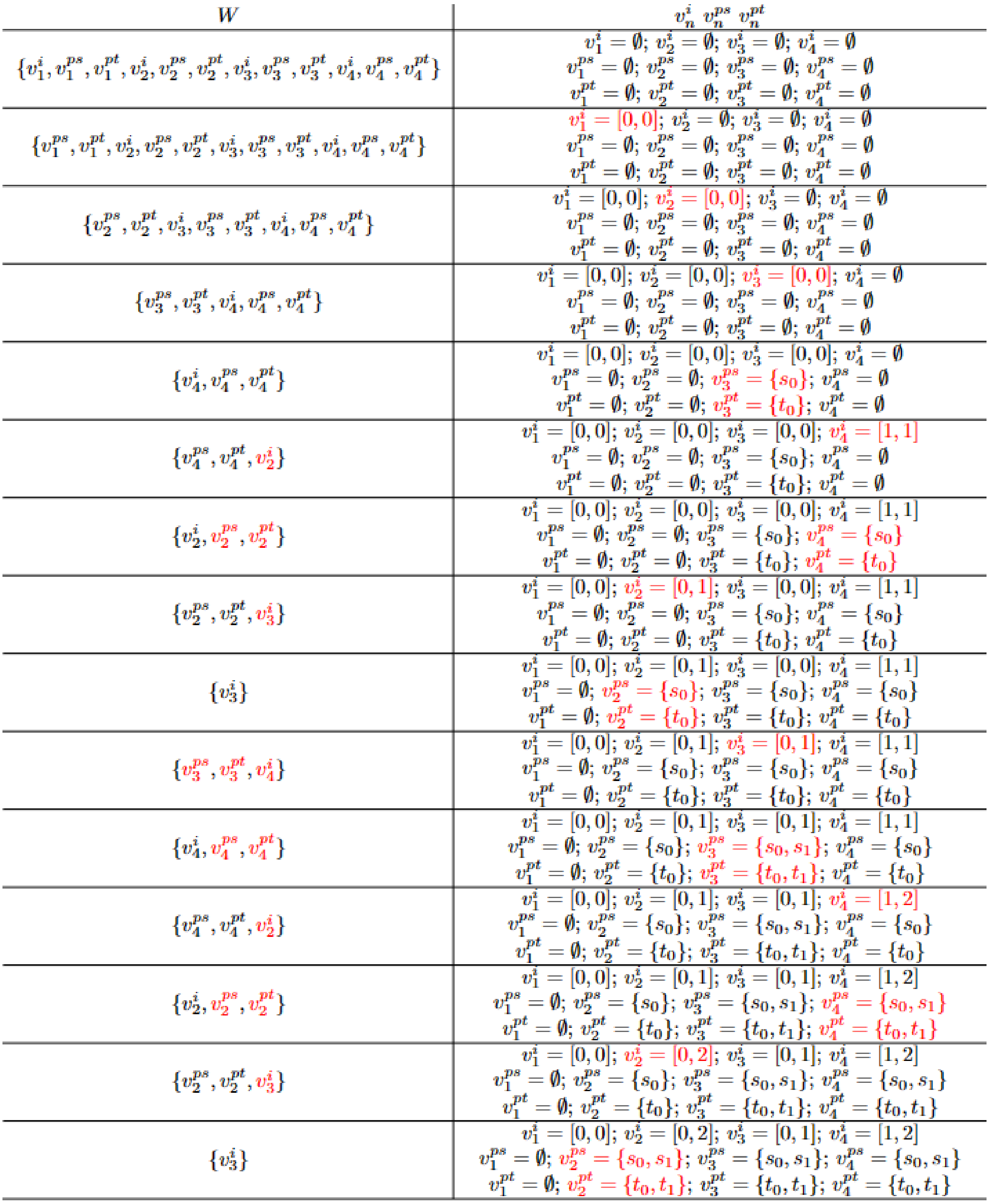
Next, we should write the formulas that detail the values for the variables i, ps, pt.

$$\begin{aligned} v_1^i &= [0,0] \\ v_2^i &= v_1^i \curlyvee v_4^i \\ v_3^i &= v_2^i \cap [-2^{31}, m] \\ v_4^i &= \{i+1 | i \in v_3^i\} \end{aligned}$$

$$\begin{aligned} v_1^{ps} &= \perp \\ v_2^{ps} &= v_1^{ps} \cup v_4^{ps} \\ v_3^{ps} &= \{s_i | i \in v_3^i\} \\ v_4^{ps} &= v_3^{ps} \end{aligned}$$

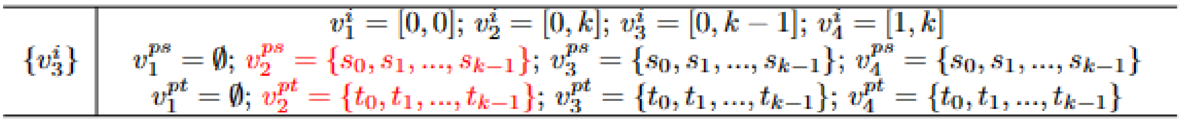
$$\begin{aligned} v_1^{pt} &= \perp \\ v_2^{pt} &= v_1^{pt} \cup v_4^{pt} \\ v_3^{pt} &= \{t_i | i \in v_3^i\} \\ v_4^{pt} &= v_3^{pt} \end{aligned}$$

We are going to use the SimpleWorkList algorithm (details in the first course) to avoid having many duplicate cells in the table. The algorithm will be applied on a generic value for m, but afterwards we will look into an example.



W	$v_n^i\ v_n^{ps}\ v_n^{pt}$
$\{v_1^i, v_1^{ps}, v_1^{pt}, v_2^i, v_2^{ps}, v_2^{pt}, v_3^i, v_3^{ps}, v_3^{pt}, v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = \emptyset;\ v_2^i = \emptyset;\ v_3^i = \emptyset;\ v_4^i = \emptyset$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \emptyset;\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \emptyset;\ v_4^{pt} = \emptyset$
$\{v_1^{ps}, v_1^{pt}, v_2^i, v_2^{ps}, v_2^{pt}, v_3^i, v_3^{ps}, v_3^{pt}, v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = [0,0];\ v_2^i = \emptyset;\ v_3^i = \emptyset;\ v_4^i = \emptyset$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \emptyset;\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \emptyset;\ v_4^{pt} = \emptyset$
$\{v_2^{ps}, v_2^{pt}, v_3^i, v_3^{ps}, v_3^{pt}, v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,0];\ v_3^i = \emptyset;\ v_4^i = \emptyset$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \emptyset;\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \emptyset;\ v_4^{pt} = \emptyset$
$\{v_3^{ps}, v_3^{pt}, v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,0];\ v_3^i = [0,0];\ v_4^i = \emptyset$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \emptyset;\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \emptyset;\ v_4^{pt} = \emptyset$
$\{v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,0];\ v_3^i = [0,0];\ v_4^i = \emptyset$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \emptyset$
$\{v_4^{ps}, v_4^{pt}, v_2^i\}$	$v_1^i = [0,0];\ v_2^i = [0,0];\ v_3^i = [0,0];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \emptyset$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \emptyset$
$\{v_2^i, v_2^{ps}, v_2^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,0];\ v_3^i = [0,0];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \{t_0\}$
$\{v_2^{ps}, v_2^{pt}, v_3^i\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,0];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \emptyset;\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \emptyset;\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \{t_0\}$
$\{v_3^i\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,0];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \{t_0\}$
$\{v_3^{ps}, v_3^{pt}, v_4^i\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,1];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0\};\ v_4^{pt} = \{t_0\}$
$\{v_4^i, v_4^{ps}, v_4^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,1];\ v_4^i = [1,1]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0, s_1\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0, t_1\};\ v_4^{pt} = \{t_0\}$
$\{v_4^{ps}, v_4^{pt}, v_2^i\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,1];\ v_4^i = [1,2]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0, s_1\};\ v_4^{ps} = \{s_0\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0, t_1\};\ v_4^{pt} = \{t_0\}$
$\{v_2^i, v_2^{ps}, v_2^{pt}\}$	$v_1^i = [0,0];\ v_2^i = [0,1];\ v_3^i = [0,1];\ v_4^i = [1,2]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0, s_1\};\ v_4^{ps} = \{s_0, s_1\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0, t_1\};\ v_4^{pt} = \{t_0, t_1\}$
$\{v_2^{ps}, v_2^{pt}, v_3^i\}$	$v_1^i = [0,0];\ v_2^i = [0,2];\ v_3^i = [0,1];\ v_4^i = [1,2]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0\};\ v_3^{ps} = \{s_0, s_1\};\ v_4^{ps} = \{s_0, s_1\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0\};\ v_3^{pt} = \{t_0, t_1\};\ v_4^{pt} = \{t_0, t_1\}$
$\{v_3^i\}$	$v_1^i = [0,0];\ v_2^i = [0,2];\ v_3^i = [0,1];\ v_4^i = [1,2]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0, s_1\};\ v_3^{ps} = \{s_0, s_1\};\ v_4^{ps} = \{s_0, s_1\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0, t_1\};\ v_3^{pt} = \{t_0, t_1\};\ v_4^{pt} = \{t_0, t_1\}$

From the table we can see that we will keep encountering the following state, where k is a natural number:



$\{v_3^i\}$	$v_1^i = [0,0];\ v_2^i = [0,k];\ v_3^i = [0,k-1];\ v_4^i = [1,k]$ $v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0, s_1, ..., s_{k-1}\};\ v_3^{ps} = \{s_0, s_1, ..., s_{k-1}\};\ v_4^{ps} = \{s_0, s_1, ..., s_{k-1}\}$ $v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0, t_1, ..., t_{k-1}\};\ v_3^{pt} = \{t_0, t_1, ..., t_{k-1}\};\ v_4^{pt} = \{t_0, t_1, ..., t_{k-1}\}$

One can see that the loop will stop when k = m+1. For that state, there will be no update to be made to v_3^i, so the list W will become empty. So, replacing in the previous formulas the value k with m+1 we will obtain the least fixed point:

$$\begin{array}{c} v_1^i = [0,0];\ v_2^i = [0, m+1];\ v_3^i = [0, m];\ v_4^i = [1, m+1] \\ v_1^{ps} = \emptyset;\ v_2^{ps} = \{s_0, s_1, ..., s_m\};\ v_3^{ps} = \{s_0, s_1, ..., s_m\};\ v_4^{ps} = \{s_0, s_1, ..., s_m\} \\ v_1^{pt} = \emptyset;\ v_2^{pt} = \{t_0, t_1, ..., t_m\};\ v_3^{pt} = \{t_0, t_1, ..., t_m\};\ v_4^{pt} = \{t_0, t_1, ..., t_m\} \end{array}$$

Now let's take, as an example m=2 and n=1. It can be seen that in the state 3, ps access the strings s_0, s_1, s_2. But we only know that s_0, s_1 are valid. This means that we are going to access s_2 which is not valid. So, a buffer overflow will occur. Going back to the generic approach, looking at state 3, it can be said that a buffer overflow will occur for all cases with m>n.

Exercise 9

Apply the meet operation for the multiplicity domain in the following cases:

a. 8·x+4·y+32·z+t=160
M(x)=2, M(y)=3, M(z)=2, M(t)=0

b. 8·x+4·y+32·z+t=90
M(x)=2, M(y)=3, M(z)=2, M(t)=2

<u>Solution</u>

Let's use the formula from the course to update the function M to a function M'.

$$M' = M\left[x_j \to \max\left(M(x_j), \min(\delta(c), \min_{i, i\neq j} \delta(a_i) + M(x_i)) - \delta(a_j)\right)\right]$$

In this formula x_j is the variable for which we currently update the multiplicity and c is the free term. We are going to apply this formula successively for each variable:
M'(x) = max(2, min(5, min(2+3,5+2,0+0)) - 3) = max(2, min(5, 0) - 3) = max(2, 0-3) = max(2, -3) = 2
M'(y) = max(3, min(5, min(3+2,5+2,0+0)) - 2) = max(3, min(5, 0) - 2) = max(3, 0-2) = max(3, -2) = 3
M'(z) = max(2, min(5, min(3+2,2+3,0+0)) - 5) = max(2, min(5, 0) - 5) = max(2, 0-5) = max(2, -5) = 2
M'(t) = max(0, min(5, min(3+2,2+3,5+2)) - 0) = max(0, min(5, 5) - 0) = max(0, 5-0) = max(0, 5) = 5

	δ(8) + M(x)	δ(4) + M(y)	δ(32) + M(z)	δ(1) + M(t)	δ(160)
M	3 + 2	2 + 3	5 + 2	0 + 0	5
M'(x)	3 + 2	2 + 3	5 + 2	0 + 0	5
M'(y)	3 + 2	2 + 3	5 + 2	0 + 0	5
M'(z)	3 + 2	2 + 3	5 + 2	0 + 0	5
M'(t)	3 + 2	2 + 3	5 + 2	0 + 5	5

In the end, the result will give the new multiplicity function M’: M’(x)=2, M’(y)=3, M’(z)=2, M’(t)=5

For exercise b, we can see from the following table that the multiplicity of all terms from the left hand side is strictly bigger than that of the right hand side member. This means that the system we are trying to solve is incompatible, so M’=⊥

	$\delta(8) + M(x)$	$\delta(4) + M(y)$	$\delta(32) + M(z)$	$\delta(1) + M(t)$	$\delta(90)$
M	3 + 2	2 + 3	5 + 2	0 + 2	1

Past year seminar exercises (optional)

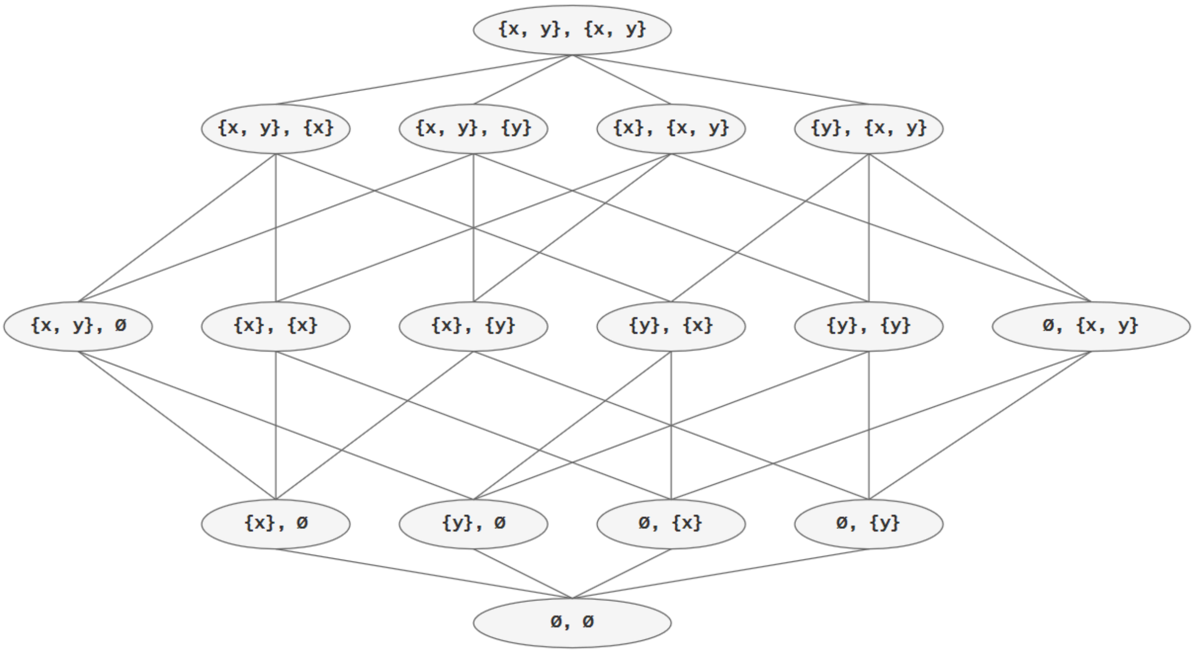
Exercise 10

Draw the lattice used for the following program:

```
x = 2;
y = 2*x;
```

Solution

As it can be seen in the solution table of a fixed point theorem from other exercises, each state of a program is associated with a lattice, precisely with the lattice of the powerset of the set of all variables from the program, denoted in the followings with L. Referring to our scenario, we have to start by defining the lattice L which is P({x,y}) since we have two variables in our program, x and y. Then the number of program states will give us the power to which we should raise the lattice L. Therefore, we obtain $L^2 = P(\{x, y\})^2$



Exercise 11

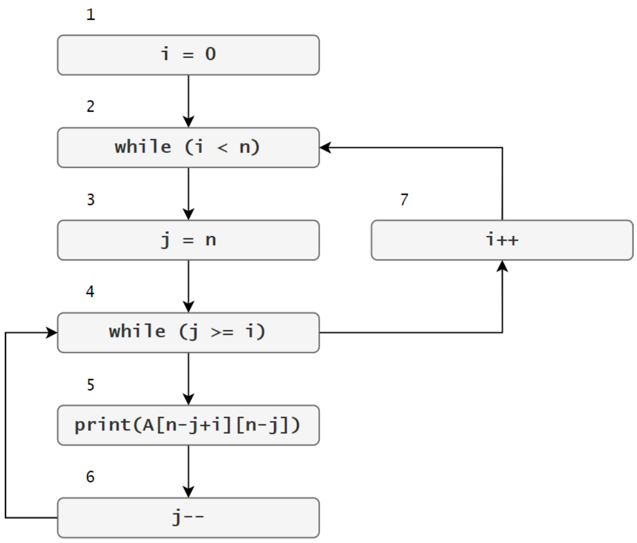
Consider that we have a square matrix A. And we want to traverse some of its diagonals. Take into consideration the following program for this task:

```
for (i=0; i<n; i++)
      for (j=n; j>=i; j--)
            print(A[n-j+i][n-j])
```

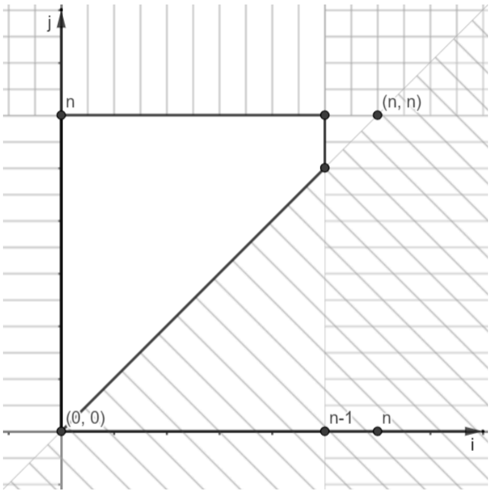
a. **Analyse if the program contains any buffer overflow vulnerability using inequalities.**
b. **Correct the program and prove that the buffer overflow is fixed.**

Solution

Let's first model the program as a CFG.



The focus will be on what happens in state 5. We can see that we can only access that state if the two loop conditions ($i<n$ and $j \geq i$) are true. Unfortunately, only using these inequalities won't be enough in our case. We are going to add two other inequations, based on empirical observations, namely $i \geq 0$ and $j \leq n$. Before drawing the inequations, we are going to turn all of them to use $\leq$ (so the semi planes will also contain the borders): $i \leq n-1$, $i \leq j$, $0 \leq i$, $j \leq n$.

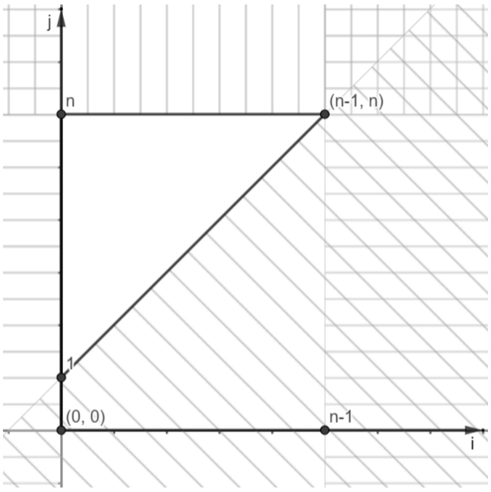


Now, we are going to look at the range of values for the line and column access: n-j, n-j+i. Specifically we are going to compute the minimum and maximum for these two functions. In a generic context, this can be done using the simplex algorithm, but we are not going to bother with that now.

min(n-j) = 0, max(n-j) = n

min(n-j+i) = 0, max(n-j+i) = n

As it can be seen, we access the line n and the column n, which will result in an overflow. To correct this, consider the inequality n-j which has the maximum n. This value should be n-1, and that points to the lower boundary for j as being the issue (we should have j>0). The inequation that gives this boundary is j≥i, which should have been j>i (precisely j≥i+1). Making the replacement, we can draw again the boundaries and recompute the minimum and maximum.



min(n-j) = 0, max(n-j) = n-1

min(n-j+i) = 0, max(n-j+i) = n-1

There is no out of bound access, so the buffer overflow was fixed.

Exercise 12

Consider the following data structures:

```
DS1:  long long arr[250];
      void* base = arr;

DS2:  struct {
            char a;
            short b[10];
            int c;
      } arr[80];
      void* base = arr;
```

a. **Tell for each case if the given addresses are valid or not regarding the alignment: *(base+23), *(base+26), *(base+48), *(base+200).**

b. **Come up with a formula that is able to describe all valid addresses for each data structure.**

Solution

Recall the length for each data type: char (1 byte), short (2 bytes), int (4 bytes) and long long (8 bytes). Using these values we can compute that for DS1 any element of the array has size 8 bytes, while elements of DS2 have size 1 + 10·2 + 4 = 25 bytes.

Since DS1 is an array of a primitive data type, there is no sense in accessing a value inside of an element (i.e. inside of a long long value), so the offsets that we access should be multiple of 8: ~~*(base+23)~~, ~~*(base+26)~~, *(base+48) = arr[6], *(base+200) = arr[25]

For DS2, we can access each element of the array, but we can also access the fields of an element. Similar to DS1, any offset that is multiple of 25 (the size of the struct) will be valid. Afterwards for each field of the struct we can compute a list of possible offsets: a (offset: 0), b[0] (offset:1), b[1] (offset:3), …, b[9] (offset: 19), c (offset: 21). So, if the offset from the base should be a multiple of 25 plus one of the offset of a field : ~~*(base+23)~~, *(base+26) = arr[1].b[0], ~~*(base+48)~~, *(base+200) = arr[8].a

Regarding the general rule of access, based on the previous observations, for DS1 we can write it as base+8·m, for any m<250. While for DS2, we can write the rules base+25·m (for a) and base+25·m+21 (for c). The rule for b, can be written applying again the rule for arrays obtaining base+25·m+1+2·n, for any m<80 and n<10.