Special Topics in Logic and Security 1

Dataflow Analysis

Paul Irofti and Ioana Leuștean

Master Year II, Sem. I, 2021-2022

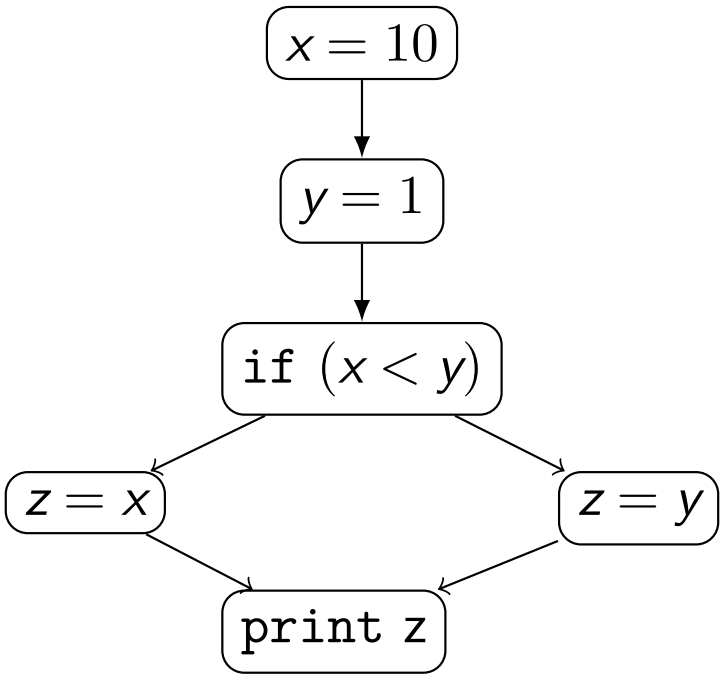
What is Static Analysis?

Static analysis involves the analysis of program properties without executing them.

Properties examples: type analysis, null pointers, unused assignments, code vulnerabilities (e.g. buffer overflow), etc.

Control Flow Graphs (CFG)

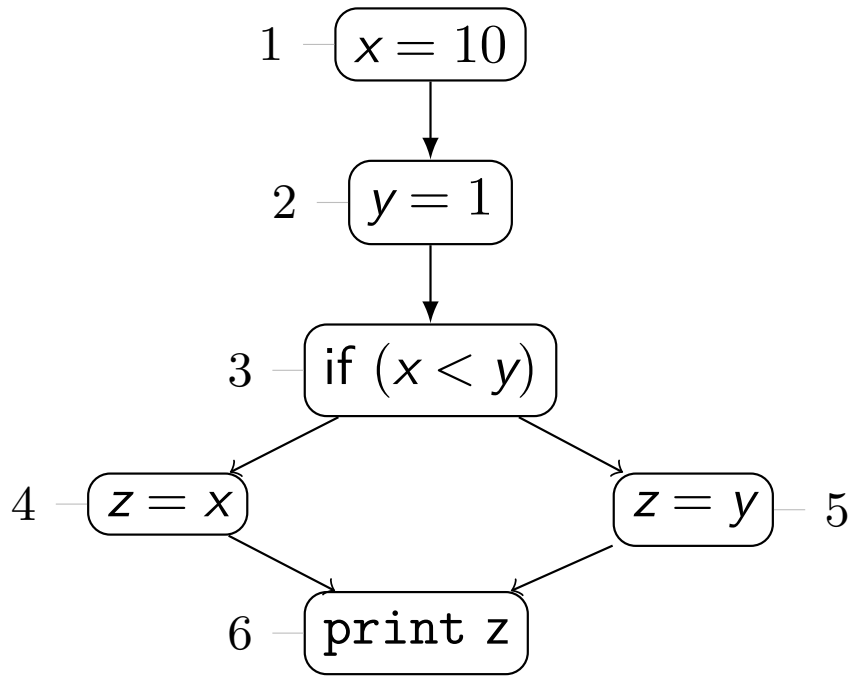
```
x = 10;
y = 1;
if  (x < y) z = x;
     else   z = y;
print z
```



Control Flow Graphs (CFG)

```
x = 10;
y = 1;
if  (x < y) z = x;
     else   z=y;
print z
```

We label each of the graph's nodes.



Example: determining live variables

https://cs420.epfl.ch/c/06_dataflow-analysis.html

A variable is active (live) at a given point in the program's execution if it is possible for the current variable value to be read before it is overwritten.

For a node n from the CFG we will denote with v_n the set of variables that are live before the execution of n.

With that

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

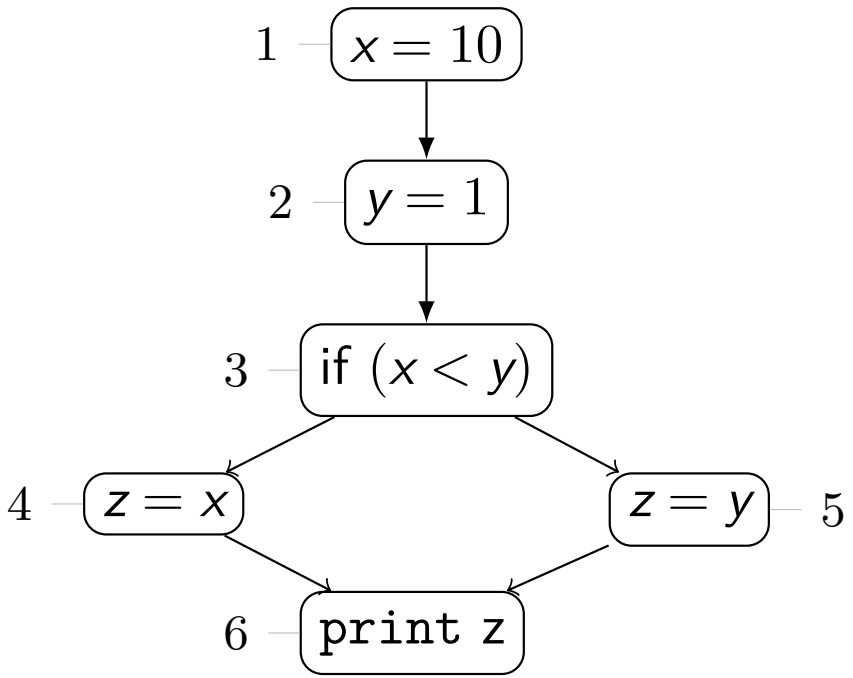
where

- $\mathit{Written}(n)$ is the set of variables that are redefined in n,
- $\mathit{Read}(n)$ is the set of variables that are read in n.

Example: determining live variables

Taking into account the above identified condition, in each node we obtain a set of constraints:

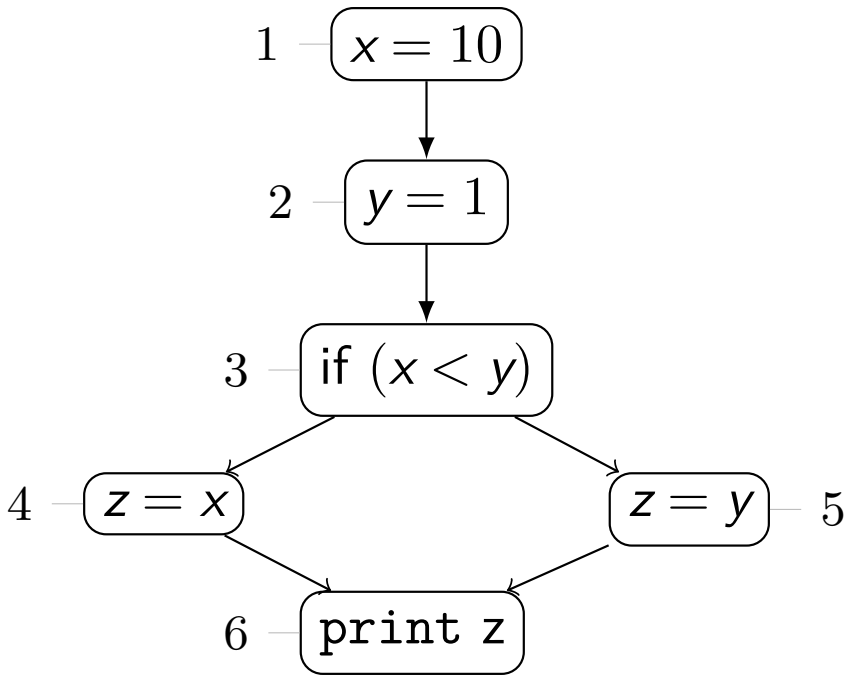
$$
\begin{array}{lcl}
v_1 & = & v_2 \setminus \{x\} \\
v_2 & = & v_3 \setminus \{y\} \\
v_3 & = & v_4 \cup v_5 \cup \{x, y\} \\
v_4 & = & (v_6 \setminus \{z\}) \cup \{x\} \\
v_5 & = & (v_6 \setminus \{z\}) \cup \{y\} \\
v_6 & = & \{z\}
\end{array}
$$



Example: determining live variables

Solving the system we obtain the solution:

$$
\begin{array}{lcl}
v_1 & = & \emptyset \\
v_2 & = & \{x\} \\
v_3 & = & \{x, y\} \\
v_4 & = & \{x\} \\
v_5 & = & \{y\} \\
v_6 & = & \{z\}
\end{array}
$$



CFG - remarks

- In the problem analysis from above, we associated a set of variables to each of the node from the control flow graph that represent the solution of a system of equations from $\mathcal{P}(Var)$, where Var is the set of variables from the entire program.

- In general, the analysis of problems from this class leads to a set of equations of latice L. Is there a generic method for solving such a system of equations?

The Product Lattice

Definition

We define the pointwise order relationship:

$$(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \iff x_i \leq y_i, \ \forall i \in \{1, \ldots, n\}$$

Theorem

If $L_1, L_2, \ldots, L_n$ are lattices then so is the product

$$L_1 \times L_2 \times \cdots \times L_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in L_i\}$$

where the lattice order $\leq$ is defined pointwise.

Remark

Let L be a lattice and $n \geq 1$, note that $(L^n, \leq)$ is a lattice. More so, if L is a complete lattice then L^n is also a complete lattice.

Equation Systems in Complete Lattices

Let L be a complete lattice and $n \geq 1$ and $F_1, \ldots, F_n : L^n \to L$ a set of monotone functions. We want to solve the equation system:

$$\begin{aligned} x_1 &= F_1(x_1, \ldots, x_n) \\ x_2 &= F_2(x_1, \ldots, x_n) \\ &\ldots \\ x_n &= F_n(x_1, \ldots, x_n) \end{aligned}$$

with $x_1, \ldots, x_n \in L$.

Definition

Denote the function $F : L^n \to L^n$ to be

$$F(x_1, \ldots, x_n) = (F_1(x_1, \ldots, x_n), \ldots, F_1(x_1, \ldots, x_n))$$

and observe that the system can be rewritten as

$$F(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$$

The equation system can be solved using the Fixed Point Theorem!

Determining live variables

It results that, in order to solve the system:

$$\begin{array}{lcl} v_1 & = & v_2 \setminus \{x\} \\ v_2 & = & v_3 \setminus \{y\} \\ v_3 & = & v_4 \cup v_5 \cup \{x, y\} \\ v_4 & = & (v_6 \setminus \{z\}) \cup \{x\} \\ v_5 & = & (v_6 \setminus \{z\}) \cup \{y\} \\ v_6 & = & \{z\} \end{array}$$

we need to find the solution to the equation:

$$F(x_1, x_2, x_3, x_4, x_5, x_6) =$$
$$(x_2 \setminus \{x\}\,,\ x_3 \setminus \{y\}\,,\ x_4 \cup x_5 \cup \{x, y\}\,,\ (x_6 \setminus \{z\}) \cup \{x\}\,,\ (x_6 \setminus \{z\}) \cup \{y\}\,,\ \{z\})$$

in the complet lattice $(\mathcal{P}(Var), \subseteq, \emptyset, Var)$.

Fixed Point Theorem Solution

$F(x_1, x_2, x_3, x_4, x_5, x_6) =$
$(x_2 \setminus \{x\}\,,\ x_3 \setminus \{y\}\,,\ x_3 \setminus \{y\}\,,\ (x_6 \setminus \{z\}) \cup \{x\}\,,\ (x_6 \setminus \{z\}) \cup \{y\}\,,\ \{z\})$

Determining the least fixed-point:

	x_1	x_2	x_3	x_4	x_5	x_6
$\bot$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$F(\bot)$	$\emptyset$	$\emptyset$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^2(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^3(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$

where $\bot = (\emptyset, \ldots, \emptyset)$

IMP and live variables

Let the set of the variables successor to the node n be:

$$Join(v_n) = \bigcup \{v_i \mid i \text{ succesor al lui } n\}$$

Then, the general formulation for live variables in node k is:

$$v_n = Join(v_n) \setminus Written(n) \cup Read(n)$$

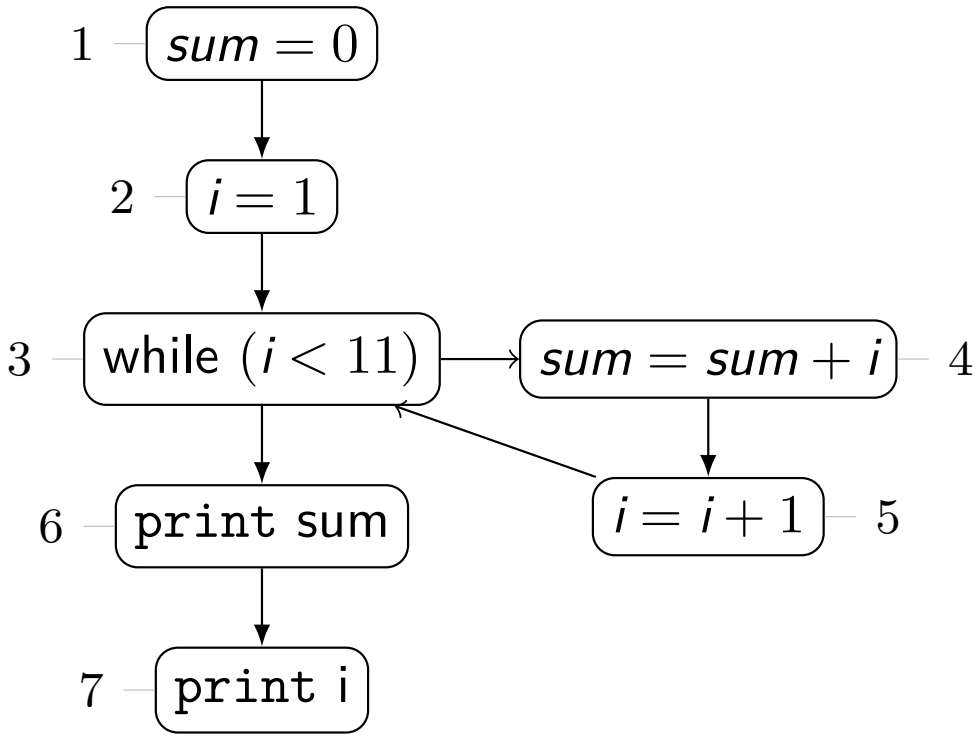
Particular formulations for IMP instructions:

- Expressions (E) — E — `x + 3, (x > 7)`
- Instruction blocks
  - Assignment — `x = E;` — $v_n = Join(v_n) \setminus \{x\} \cup Var(E)$
  - Conditional — `if (E)` — $v_n = Join(v_n) \cup Var(E)$
  - Loops — `while (E)` — $v_n = Join(v_n) \cup Var(E)$
  - Output — `print E;` — $v_n = Join(v_n) \cup Var(E)$

where $Var(E)$ represents the set of variables present in expression E.

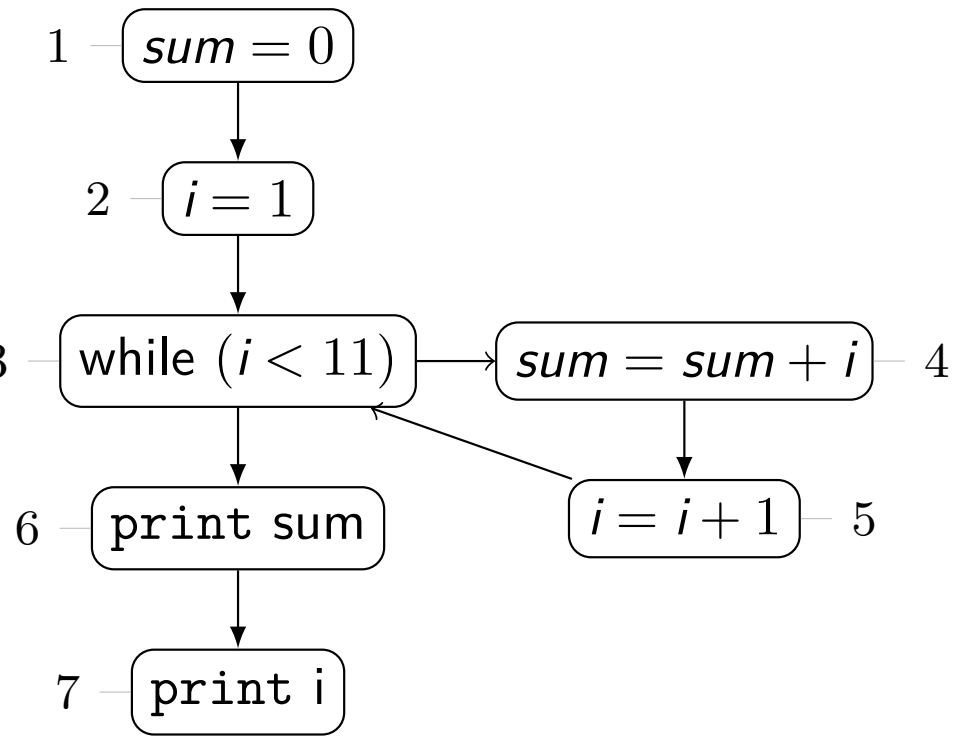
Control Flow Graphs (CFG)

```
sum = 0;
i = 1;
while (i < 11) {
    sum = sum + i;
    i = i + 1;
}
print sum
print i
```



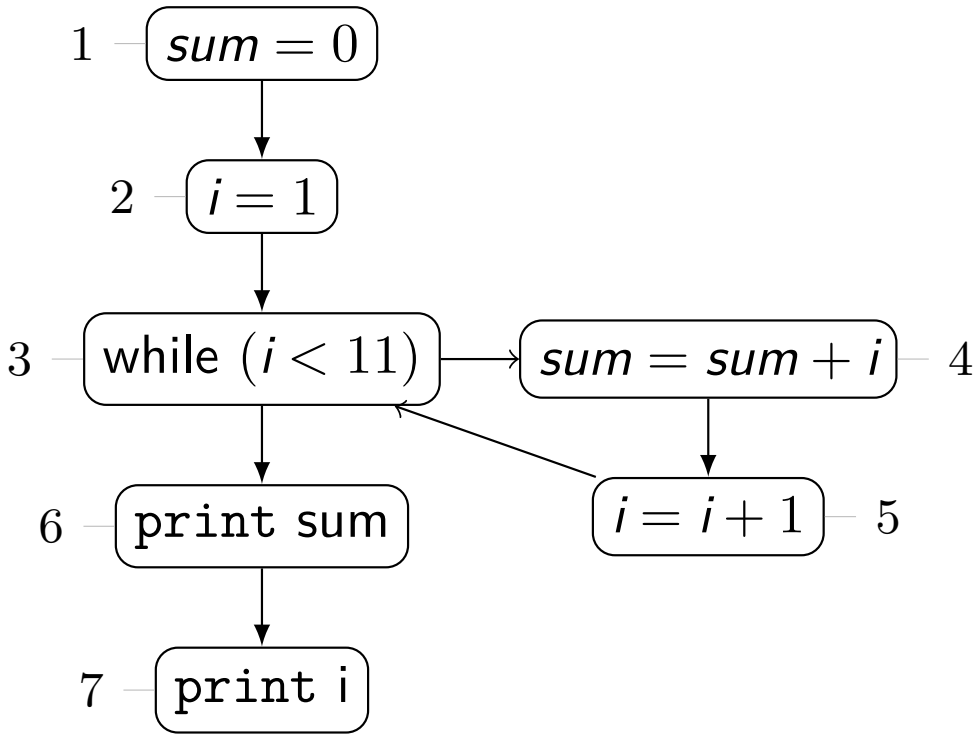
Control Flow Graphs (CFG)

$$
\begin{aligned}
v_1 &= v_2 \setminus \{sum\} \\
v_2 &= v_3 \setminus \{i\} \\
v_3 &= v_4 \cup v_6 \cup \{i\} \\
v_4 &= (v_5 \setminus \{sum\}) \cup \{i, sum\} \\
v_5 &= (v_3 \setminus \{i\}) \cup \{i\} \\
v_6 &= v_7 \cup \{sum\} \\
v_7 &= \{i\}
\end{aligned}
$$



Control Flow Graphs (CFG)

$$
\begin{aligned}
v_1 &= \emptyset \\
v_2 &= \{sum\} \\
v_3 &= \{i, sum\} \\
v_4 &= \{i, sum\} \\
v_5 &= \{i, sum\} \\
v_6 &= \{i, sum\} \\
v_7 &= \{i\}
\end{aligned}
$$



Solving with the Fixed Point Theorem

$F(x_1, x_2, x_3, x_4, x_5, x_6, x_7) =$
$(x_2 \setminus \{sum\}, x_3 \setminus \{i\}, x_4 \cup x_6 \cup \{i\}, (x_5 \setminus \{sum\}) \cup \{i, sum\}, x_3 \setminus \{i\}) \cup \{i\}, x_7 \cup \{sum\}, \{i\})$

Determining the least fixed-point:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
$\bot$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$F(\bot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{sum\}$	$\{i\}$
$F^2(\bot)$	$\emptyset$	$\emptyset$	$\{sum, i\}$	$\{sum, i\}$	$\emptyset$	$\{sum\}$	$\{i\}$
$F^3(\bot)$	$\emptyset$	$\{sum\}$	$\{sum, i\}$	$\{sum, i\}$	$\{sum, i\}$	$\{sum\}$	$\{i\}$

where $\bot = (\emptyset, \ldots, \emptyset)$

Exercise

Perform liveness analysis for the following program:

```
x = 42;
while (x > 1) {
    y = x / 2;
    if (y > 3)
        x = x - y;
    z = x - 4;
    if (z > 0)
        x = x / 2;
    z = z - 1;
}
print x
```

Algorithms for determining the *lfp*

In order to obtain the least-fixed point (*lfp*) we use the following chain until we converge:

$$\mathbf{F}^0(\bot) = \bot \leq \mathbf{F}(\bot) \leq \mathbf{F}^2(\bot) \leq \cdots \leq \mathbf{F}^n(\bot) \leq \cdots$$

Which can be formalized in pseudo-code as:

Algorithm 1: NaiveLFP

```
1 x = (⊥, ⊥, ..., ⊥);
2 while x ≠ F(x) do
3     x = F(x);
4 return x;
```

Complexity: depends on the height of lattice L^n and the evaluation cost of $\boldsymbol{F}(\boldsymbol{x})$.

Convergence: requires n iterations.

Algorithms for determining the *lfp*

Algorithm 1 does not profit from the lattice structure:

- recomputes the inputs that were not changed in the former iteration
- does not take into consideration the inputs that were already computed in the current iteration

In the first example we had:

$$F(x_1, x_2, x_3, x_4, x_5, x_6) =$$
$$(x_2 \setminus \{x\}\,,\ x_3 \setminus \{y\}\,,\ x_4 \cup x_5 \cup \{x, y\}\,,\ (x_6 \setminus \{z\}) \cup \{x\}\,,\ (x_6 \setminus \{z\}) \cup \{y\}\,,\ \{z\})$$

and the fixed-point is computed in 6 iterations

	x_1	x_2	x_3	x_4	x_5	x_6
$\bot$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$F(\bot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{z\}$
$F^2(\bot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{x\}$	$\{y\}$	$\{z\}$
$F^3(\bot)$	$\emptyset$	$\emptyset$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^4(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^5(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$

Comparisson of *lfp* algorithms

Ideally, an efficient algorithm would reduce the number of iterations:

	x_1	x_2	x_3	x_4	x_5	x_6
$\bot$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$F(\bot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{z\}$
$F^2(\bot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{x\}$	$\{y\}$	$\{z\}$
$F^3(\bot)$	$\emptyset$	$\emptyset$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^4(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^5(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$

versus

	x_1	x_2	x_3	x_4	x_5	x_6
$\bot$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$F(\bot)$	$\emptyset$	$\emptyset$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^2(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$
$F^3(\bot)$	$\emptyset$	$\{x\}$	$\{x, y\}$	$\{x\}$	$\{y\}$	$\{z\}$

Round Robin *lfp*

Gauss-Seidel-like component-wise updates:

- computes x_i based on $x_{j<i}$ from the current iteration
- *lfp* is attained even though the outer iteration may provide a different output than the outer iteration in Alg. 1

Algorithm 2: RoundRobinLFP

```
1 (x_1, x_2, ..., x_n) = (⊥, ⊥, ..., ⊥);
2 while (x_1, x_2, ..., x_n) ≠ F(x_1, x_2, ..., x_n) do
3     for i ∈ 1 → n do
4         x_i = F_i(x_1, x_2, ..., x_n);
5 return (x_1, x_2, ..., x_n);
```

Complexity: depends on the height of lattice L^n and the cost of evaluating $\boldsymbol{F}_i(\boldsymbol{x})$.

Convergence: requires at most n iterations.

Chaotic *lfp*

Stochastic component wise updates:

- the order of operations is not important in Alg. 2
- computes x_i based on the x_j's from the current iteration; in no particular order
- Alg. 2 still recomputes the unchanged inputs from the former iteration
- the constraints need to be applied until convergence

Algorithm 3: ChaoticLFP

1 $(x_1, x_2, \ldots, x_n) = (\bot, \bot, \ldots, \bot)$;
2 **while** $(x_1, x_2, \ldots, x_n) \neq \boldsymbol{F}(x_1, x_2, \ldots, x_n)$ **do**
3 $\quad i = \mathit{random}(1 \rightarrow n)$;
4 $\quad x_i = \boldsymbol{F}_i(x_1, x_2, \ldots, x_n)$;
5 return $(x_1, x_2, \ldots, x_n)$;

Complexity: first, it depends on how i is chosen at each iteartion, then it depends on the height of lattice L^n and then on the cost of evaluating $\boldsymbol{F}_i(\boldsymbol{x})$.

Convergence: the algorithm is not guaranteed to stop, but if it does the result is correct.

SimpleWorkList *lfp*

Define the map $dep : Nodes \to 2^{Nodes}$, where $dep(v)$ is the set of nodes whose information depends on the information of node v.

Algorithm 4: SimpleWorkListLFP

```
1  (x_1, x_2, ..., x_n) = (⊥, ⊥, ..., ⊥);
2  W = {v_1, v_2, ..., v_n};
3  while W ≠ ∅ do
4      v_i = W.removeNext();
5      y = F_i(x_1, x_2, ..., x_n);
6      if y ≠ x_i then
7          x_i = y;
8          for v_j ∈ dep(v_i) do
9              W.add(v_j);
10 return (x_1, x_2, ..., x_n);
```

The functions *removeNext*() and *add*() choose a random node from W and add a node in W, respectively.

SimpleWorkList *lfp*

Properties

- computes only the particular constraints for the current node
- does not recompute the inputs unchanged during the former iteration
- computes x_i based on the x_j's from the current iteration; in no particular order
- constraints are applied until convergence
- each iteration has the same effect as the Alg. 3 iteration

Complexity: depends on the number of nodes n, the lattice height h, and the cost of evaluating $\boldsymbol{F}_i(\boldsymbol{x})$.

Convergence: a single iteration either decreases the work volume in W or climbs up the lattice; the algorithm stops in finite time because the lattice height is finite and the iterations stop when W is void.