Program Slicing

↪ Extract an executable subset of a program that (potentially) affects that value at a particular program location.

↪ Slicing Criterion ⇒ Program location + Variable.

Eg:-

```
1  n = int(input("enter a number"))
2  i = 1
3  sum = 0
4  prod = 1
5  while i <= n:
6          sum += i
7          prod *= i
8          i += 1
9  print(sum)   <-- Slice for value of sum at this statement.
10 print(prod)
```

Backward ⇒ Statements that influence the Slicing Criterion

Forward ⇒ " " are " by the Slicing Criterion.

Applications
- ↪ Debugging
- ↪ program understanding
- ↪ Change impact analysis
- ↪ parallelization.

(14)

Program Dependence Graph

It is directed graph representing data & control dependences.

E ⇒ Data flow dependences / Control " "

N ⇒ Statements.

Variable Definition & variable use

A VD for a variable V is a B.B. that assigns to V

A VU for a variable V is a BB that reads the value of V

Definition - Clear paths

A DCP for a variable V is a path $n_1 \ldots n_k$ in the CFG such that

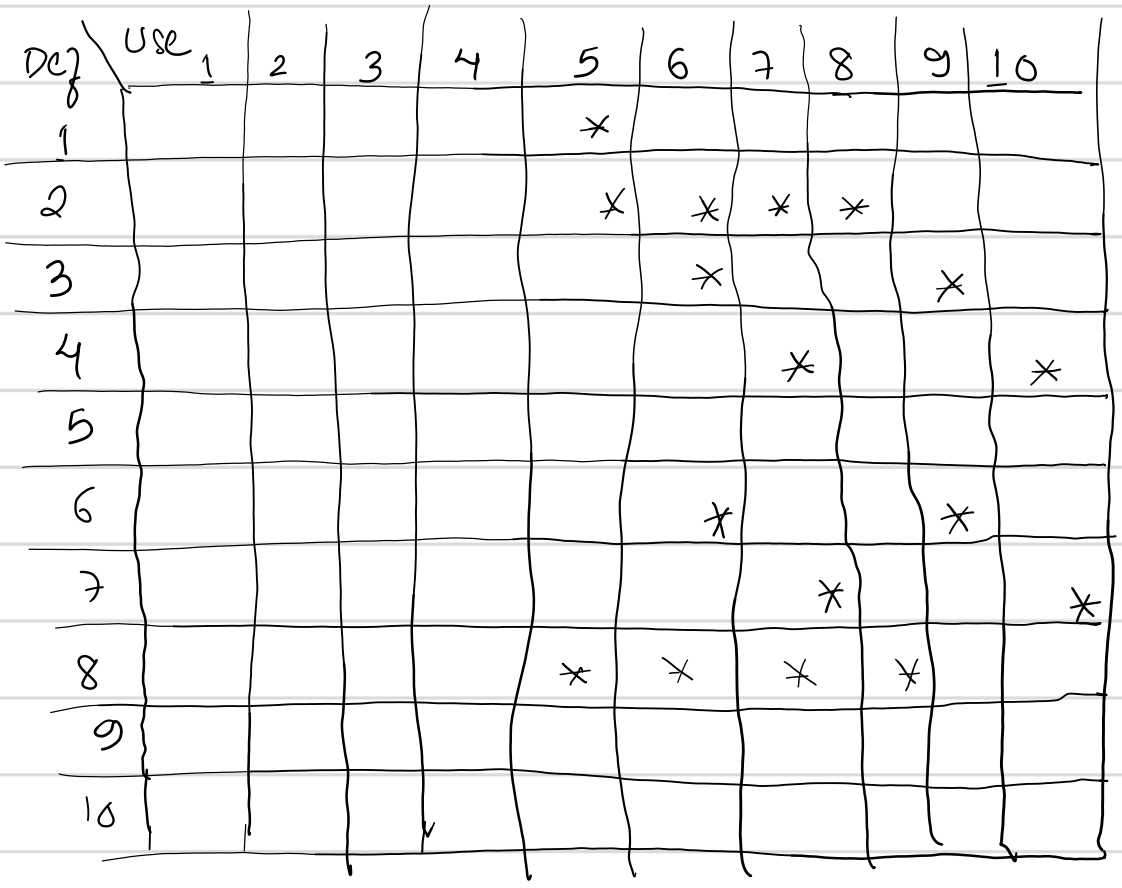
n_1 is variable definition for V

n_k " variable use for V

Definition - use pair (DU pair)

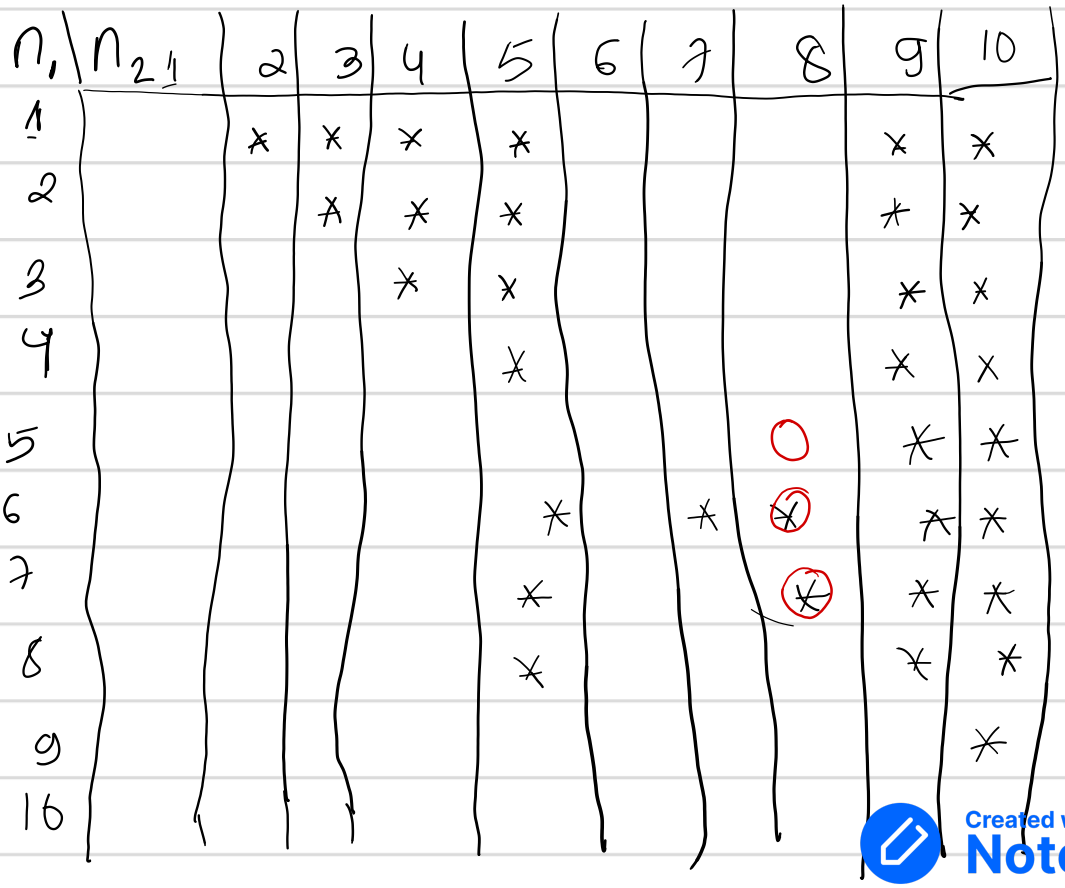
- It is a variable U is a pair of nodes (d, u) such that there is a definition path d .... u in CFG.

Data flow Dependencies



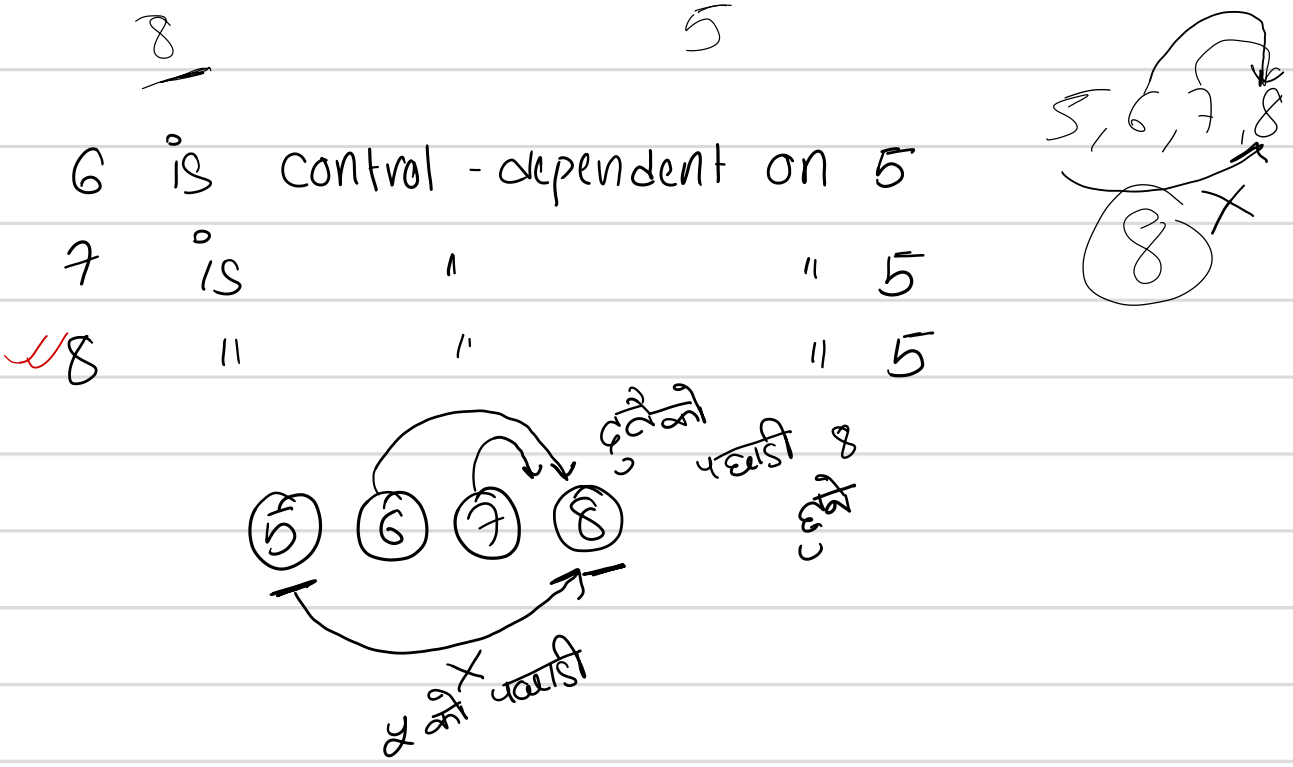
Control flow dependences

① Strict post dominators



Control Flow Dependences

- Post-dominator:
  Node n_2 (strictly) post-dominates node $n_1 (\neq n_2)$ if every path $n_1, ..., exit$ in the control flow graph contains n_2
- Control dependence:
  Node n_2 is control-dependent on node $n_1 \neq n_2$ if
  - there exists a control flow path $P = n_1, ..., n_2$ where n_2 post-dominates any node in P (excluding n_1), and
  - n_2 does not post-dominate n_1



Computing Slices

Given:

- Program dependence graph G_{PD}
- Slicing criterion (n, V), where n is a statement and V is the set of variables defined or used at n

Slice for (n, V):

All statements from which n is reachable
(i.e., all statements on which n depends)

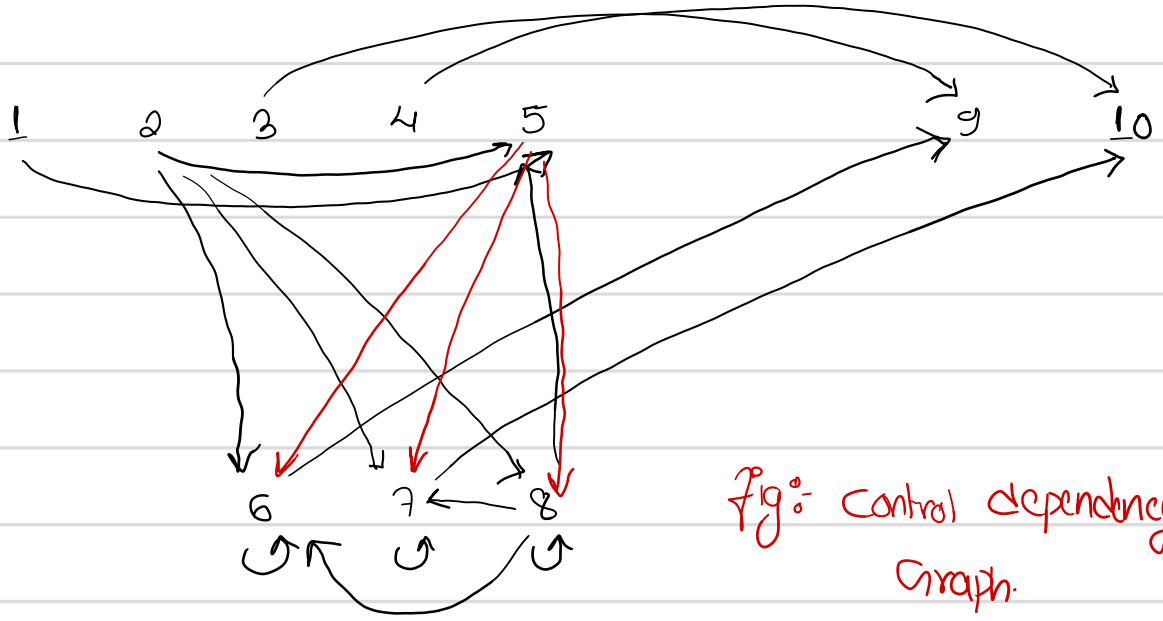


Fig: Control dependency Graph.

→ Data dependency.

→ Control dependency

Slice(9, {sum})
→ {n | reachable (n, 9)}
→ {1, 2, 3, 5, 6, 8, 9}

```
var x = 1;      // 1
var y = 2;      // 2
if (x < y) {    // 3
  y = x;        // 4
}
var z = x;      // 5
```

Draw the PDG and compute $slice(5, \{z\})$.
What is the sum of

- **the number of nodes,**
- **the number of edges, and**
- **the number of statements in the slice?**

Symbolic Execution :-

↳ White box (looking into program).

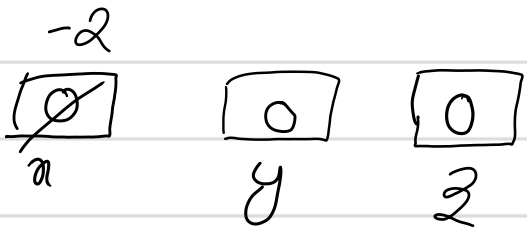
↳ Reasons about behaviour of program by "executing" it with Symbolic values.

Ex:-

```
def fun(a, b, c):
    x = y = z = 0
    if a:
        x = -2
    if b > 5:
        if !a and c:
            y = 1
        z = 2
    assert x + y + z != 3
```

Concrete Execution

$a = b = c = 1$.



assert $-2 + 0 + 0 \neq 3$ ✓

passed.

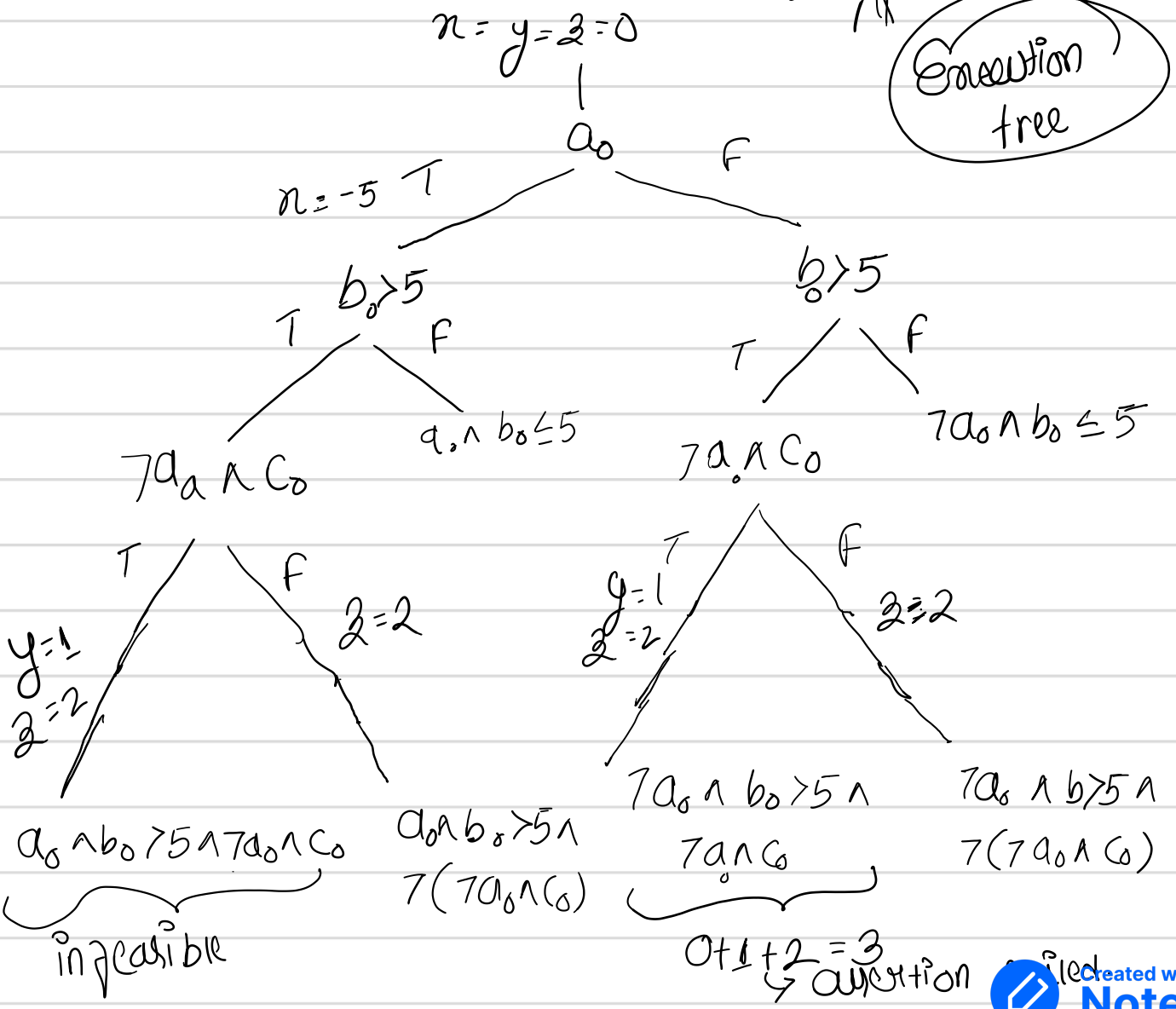
Symbolic Execution

$a = a_0$, $b = b_0$, $c = c_0$ ← Initial values.

↝ Symbolic values.

Binary tree.

Execution tree



Dynamic Program Analysis.

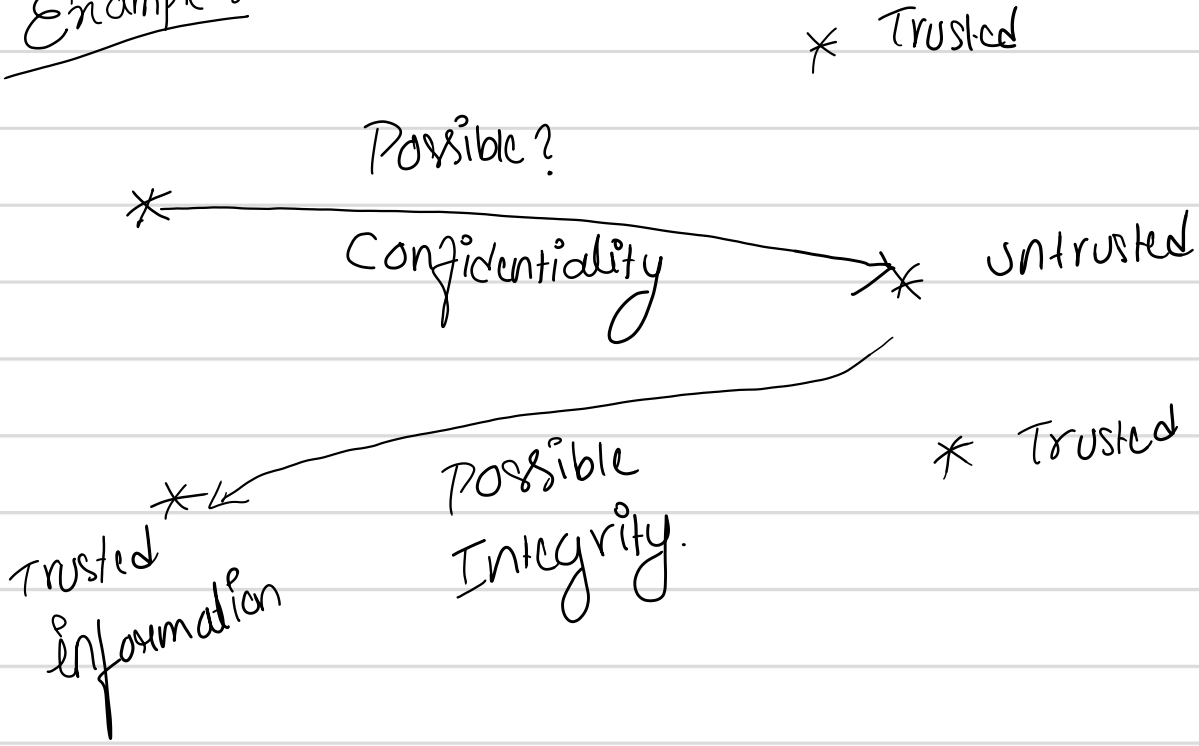
a) Information flow analysis
b) Dynamic Program Slicing.
c) Test generation
d) Path profiling.

a) Information Flow Analysis:

<) Secure the data manipulated by a computing system.

→ Confidentiality & Integrity.
→ Check whether information from one code location propagates to another location.

Example:



Ex) Confidentiality

```
CreditCard = 5678
x = creditcard
visible = False
if x > 1000:
    visible = True
```

← Secret information propagates to x

↑ Secret information (partly) propagates to visible.

Ex Integrity

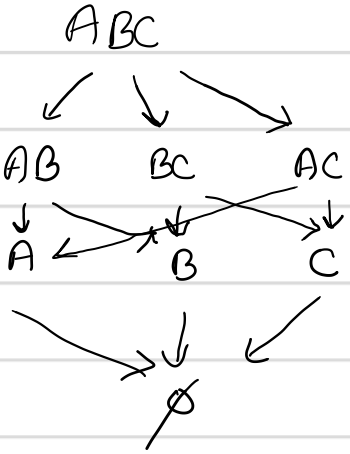
```
designatedPresident = "Ram"
x = input()
designatedPrecident = x
```

↑ low-integrity info propagates to high-integrity.

lattice of Security levels

High
↓
low

Top Secret
↓
Secret
↓
Confidential
↓
Public



Universally Bounded lattice

Tuple $(S, \rightarrow, \perp, \top, \oplus, \otimes)$

- S → set of security classes
- $\rightarrow$ → Partial order
- $\perp$ → least upper bound
- $\top$ → greatest lower bound.

$S \rightarrow \{ABC, AB, AC, BC, A, B, C, \emptyset\}$

→ partial order (more secret to less secret)

$\perp$ → lower bound → $\emptyset$

$\top$ → upper bound → ABC

$\oplus$ → least upper bound operator → combine, union

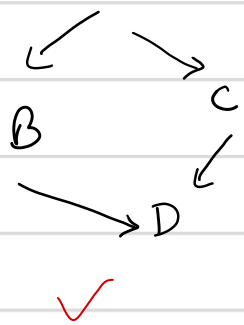
$\otimes$ → greatest lower bound " → intersection

$AB \oplus A = AB$ $\qquad$ $ABC \otimes C = C$

$\emptyset \oplus AC = AC$

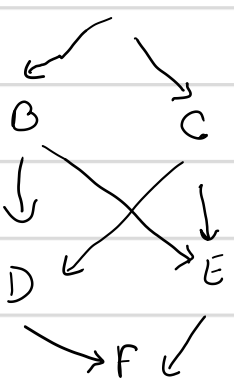
Q. Is the following U.B.L?

(1)



✓

(2)



$D \oplus E = ?$

$= \{A, B, C\}$

→ no least upper bound.

Information flow Policy :-

(*) No information flow from source to sink.

Explicit flow => Caused by data flow dependency
Implicit " => " " control " "

```
x = CreditCardNb          if x > 1000:
```

Explicit flow.

Implicit flow

Dynamic information flow analysis

- Associate Security labels with memory locations.
- propagate labels at runtime.

Taint Sources & Sinks

possible source => variables, return values
possible sinks => variables, parameters.

Taint Propagation :-

(1) Explicit flows.

label(result) ← label(inp$_1$) ⊕ .... ⊕ label(inp$_2$)

(2) Implict flows.

maintains stack.

push the label.

∴ Ex.

label(CreditCardNb) = Secret

CreditCardNb = 5678

label(x) = Secret

x = CreditCardNb

label(visible) = Public

visible = False

if x > 1000:

produce intermediate value of b.

label(b) = label(x) ⊕ label(1000)
= secret.

Push "secret" onto Stack.

visible = True

policy
- Security classes : Public, Secret
- Source : variable "creditCardNb"
- Sink : variable "visible"

Secret ↓ Public

Stack

labels on ^ are part of input

label(visible) = Secret ⊕ label(true)
= Secret ⊕ public
= Secret
⇒ violation of policy.

x = getX() ①

y = x + 5 ②

z = True ③

if x == 10: ④

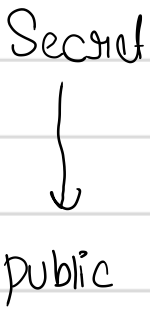
z = False ⑤

foo(z) ⑥

Suppose that getx returns 5. write down the labels after each operation. Is there a policy violation.

policy → Security classes Public & Secret.

Security lattice



LUB (P,S) = S
LUB (P,P) = P
LUB (S,S) = S

label(x) = Secret ①

label(y) = label(x) ⊕ label(5) ②
= Secret ⊕ public
= Secret.

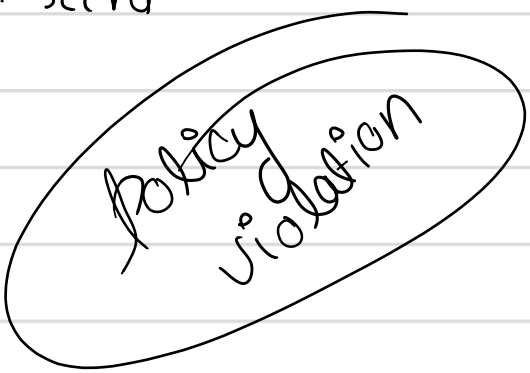
label(z) = label(True) = public ③

label(b) = label(x) ⊕ label(10) ④
= Secret ⊕ public
= Secret.

push Secret to Stack.

label(z) = label(false) ⊕ secret ⑤
= Public ⊕ secret
= Secret.

label(z) = Secret ⑥



Dynamic Slicing

↳ Statements of an execution that must be executed to give a variable a particular value.

↳ Slice for one i/p different from slice of another i/p.

a) Execution history → Sequence of PDG.

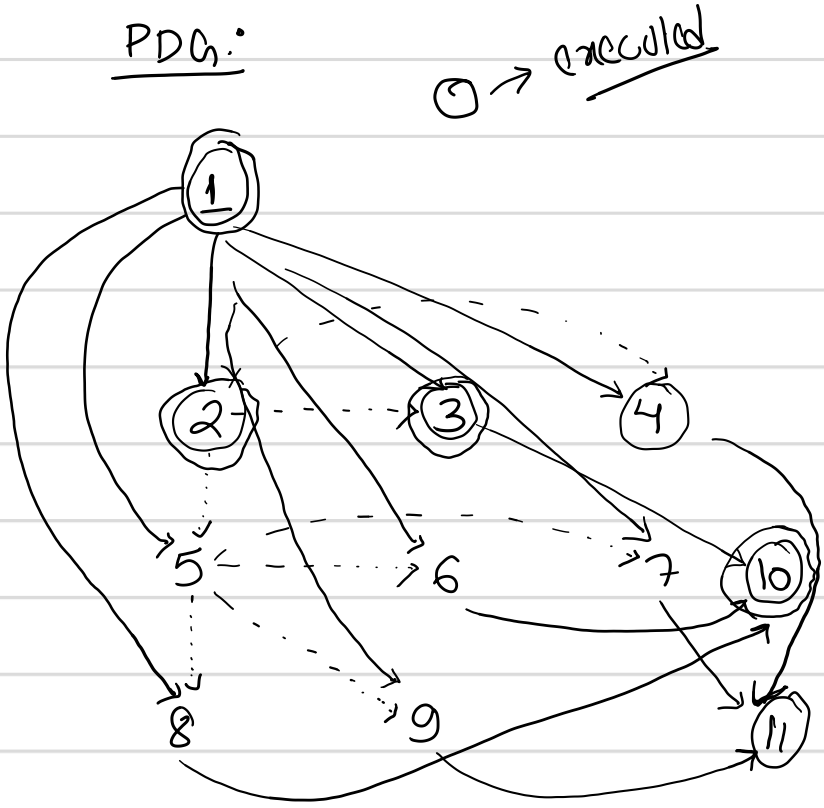
b) Slice for statement n & variable v:

```
1   x = input()
2   if x < 0:
3       y = x+1
4       z = x+2
    else:
5       if x == 0:
6           y = x+3
7           z = x+4
        else:
8           y = x+5
9           z = x+6
10  print(y)
11  print(z)
```

Input = -1

History : 1, 2, 3, 4, 10, 11

PDG:



Slice (10, {y})

⇒ { 1, 2, 3, 10 }.

Iizna

Consider the code

```
n = input()
z = 0
y = 0
i = 1
while i <= n :
    z = z + y
    y = y + 1
    i = i + 1
print(z)
```

for n = 1

Q1. Use method discuss in the lecture. Discuss the limitation.

Q2. Is there more accurate method? Explain.

Random Testing & Fuzzing :

① Black Box
② Grey Box
③ White Box.

} Use feedbacks