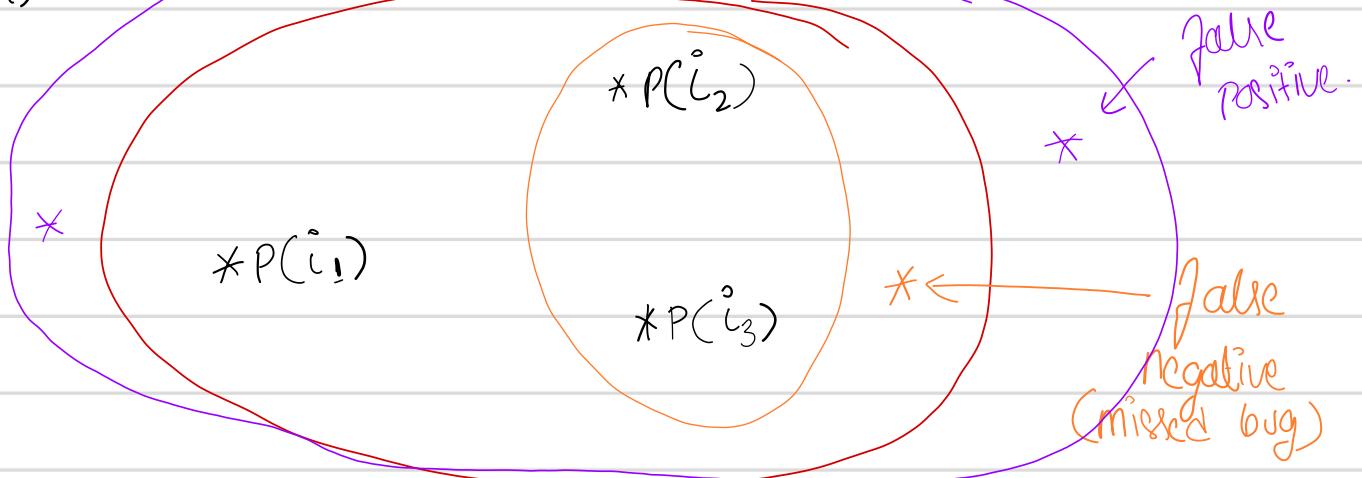


Program Analysis

Let Program P, input i and behaviour of program

$P(i)$



All possible behaviours (This is what we want ideally)

Underapproximation (Example :- testing, dynamic analysis)

Overapproximation (most static analysis)

$$FP \Rightarrow \text{El } \pi \text{ is } \sqrt{2} \text{ is } 1$$

$$FN \Rightarrow \text{an } \pi \text{ is } \sqrt{2} \text{ is } 1$$

Foundation of PA :-

Programming language = Syntax + Semantic + Implementation

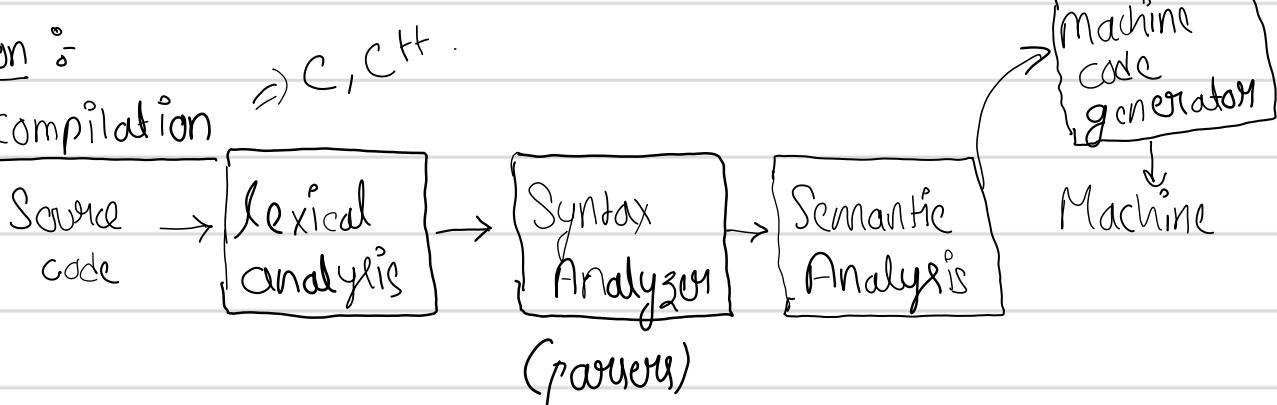


- how do they look like
- what characters are written (rules)
- meaning
- what will happen
- Execution



Implementation :-

a) Compilation



phases of compilation

- b) Interpretation ⇒ Executing a program directly from the source code without first translating it into machine code.
- c) Hybrid
 ↳ Java, JavaScript
 ↳ Python.
 ⇒ Python.

Syntax :-

a) Grammar.

Arithmetic expression

Terminal $\Sigma = \{ 0, 1, 2, \dots, 9, +, - \}$

Nonterminal $N = \{ EXP, NUM, OP, DIGIT \}$

$S = EXP$

$P = EXP \rightarrow NUM \mid EXP \ OP \ EXP$

$OP \rightarrow + \mid -$

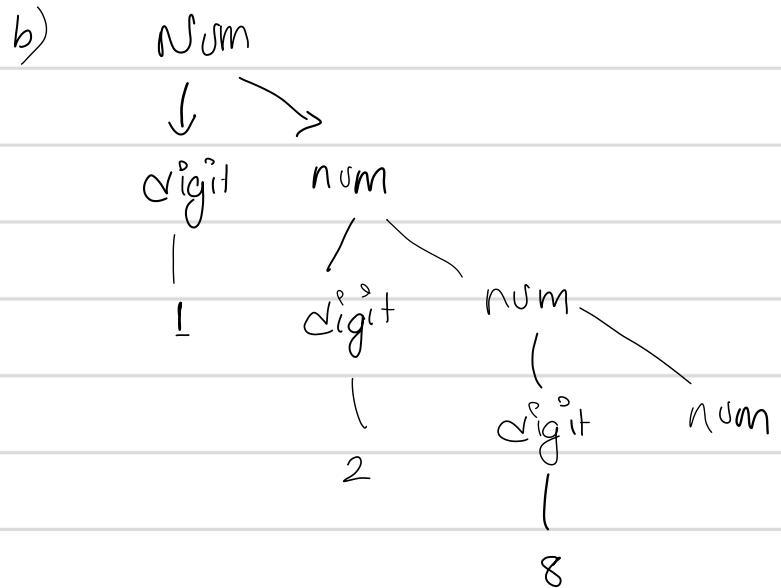
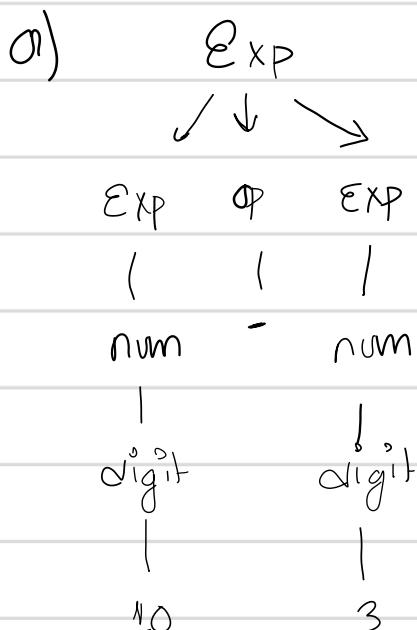
$NUM \rightarrow DIGIT \mid DIGIT \ NUM$

$DIGIT \rightarrow 0 \mid 1 \mid 2 \dots \mid 9$

Q. What is the part of language ??

- a) $10 - 3$ ✓
- b) $11 * 4$ ✗
- c) 128463 ✓
- d) $10 + (4 - 3)$ ✗





④ Abstract Syntax Tree :-

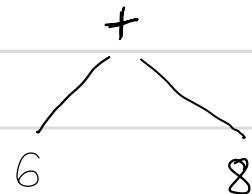
Abstract Grammar.

Eg:- $E \rightarrow n \mid op \mid E, E)$
 $op \rightarrow + / -$

- terminals are the tokens.

Example :-

6 + 8



⑤ Control Flow Graphs,

→ Models the flow of control through the program.

$$G = (N, E)$$

where, Nodes are basic blocks

(

Graph is made of Node & Edge.

(Sequence of operations executed together)

E possible transition of

Example:-

①

if $y > 5$:
 $x = 5$

else:

$x = 7$
print (x)

if $y > 5$

$x = 5$

$x = 7$

Print(x)

② while $y > 5$:

$x = x + 1$

$y = x$

print (x)

while $y > 5$

$x = x + 1$
 $y = x$

Print(x)

③ Control flow graph focuses on the order of the execution of the program.

Data dependency graph :-

- It models the flow of data from "definition" to "use"

$$G = (N, E)$$

$\circ N$ is operations
 E is possible def-use relationship

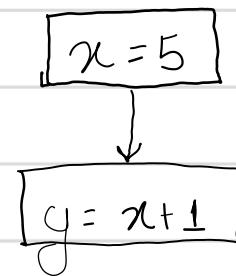
$E = (n_1, n_2)$ means : at node n_2 , some data may be used that is defined at n_1



Ex 1

$$x = 5$$

$$y = x + 1$$



Ex 2

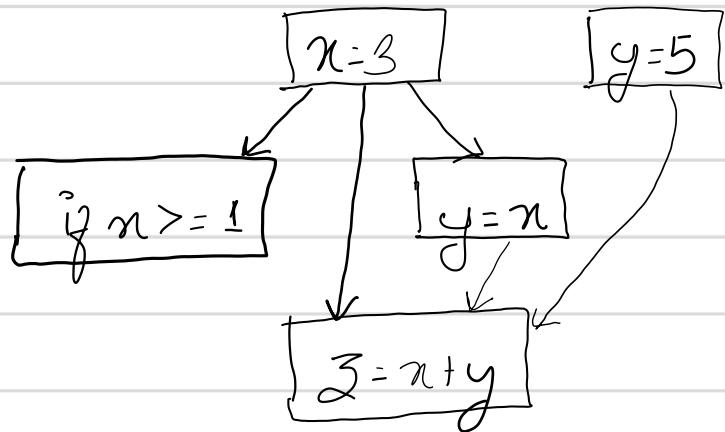
$$x = 3$$

$$y = 5$$

if $n \geq 1$:

$$y = n$$

$$z = x + y$$



④ Ingredients of PA :-

① Semantic Properties & Syntactic Properties .

⇒ Semantic property can be completely defined w.r.t the set of executions of a program, while a Syntactic property can be decided directly based on the program text.

which properties.

- ① A program may print text to the console
- ② A program may call print
- ③ A program prints to console.
- ④ A program consists of one line of code.



→ 1, 2 & 4 are Syntactic properties
→ 3 is Semantic property.

If $x = 1$:

```
print("Hello")
```

Suppose this is the code and here are the result.

④ Rice's Theorem:

- It states that all non-trivial Semantic properties of program are undecidable.
- A property is non-trivial if it is neither true for every partial computable function, nor false for every partial computable function.

Proof:

Halting Problem is a problem of determining from

1) a description of an arbitrary computer program

2) an input

whether the program will finish running or continue to run forever. A general algorithm to solve problem for all possible input pairs cannot exists.

We can make approximation



Non-trivial Properties

- A program exists
 - A program prints "Hello"
 - A program finished in less than 5 seconds
 - A program dies with "Segmentation Fault"
- Functionality ↗
- Security bug ↗
- A program prints user password to the console.

Nontrivial Properties

- A functional property about programs is **nontrivial** if some programs have the property and some do not
 - Program never goes into an infinite loop **nontrivial**
 - Program accepts a finite number of strings **nontrivial**
 - Program accepts 0 or more inputs **trivial**

Static Analysis :-

```
def fun():
    n = 0
    return 42/n
```

Bug : yes

```
def fun():
    return 42/n
```

Bug : no



C⁺⁺ → clang Tidy

Python → Pyright

Rice Theorem \Rightarrow no 100% sure.

Style checking :-

C⁺⁺ → CppLint.

Python → Pylint.

```

int f (int n) {
    return 42/n;
}
    
```

C⁺⁺ int f (int n){
 return 42/n;
}

→ Extra Space before C function call.
Should be at end of previous line

① Dynamic Analysis is testing [AddressSanitizer]

Consider the following Program.

```

def fun(n):
    list = [1, 2, 3, 4]
    return list[n]
    
```

→ Pyright

→ Mypy

→ Bandit

print(fun(5))

ERROR!

Traceback (most recent call last):

File "<main.py>", line 7, in <module>

File "<main.py>", line 5, in fun

IndexError: list index out of range

==== Code Exited With Errors ===

Index error :-

List index out of range

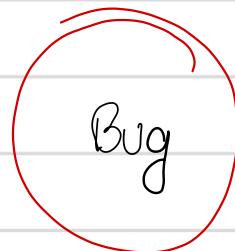


Quality of Analysis :-

Sound & Complete

Incomplete

1



False Negative

2



3



4



"Program में Bug
Report वर्ता Analyzer से
होना सकता है"

False positive

Unsound.

"Program में Bug छिन नहीं
Analyzer से दे सकती"

Q. Analyze the following code samples using Pylint, pyright and prepare the report

script1.py

import os

```
def divide(a: int, b: int) -> float:  
    return a / b
```

```
result = divide(10, 0)
```



1. 10 → Bug

script2.py

```
def greeting(name: str) -> int:  
    return f"Hello, {name}!"
```

2. yes → Bug

```
message = greeting("World")  
print(message)
```

3. yes → Bug

4. yes → No Bug.

script3.py

```
def add_numbers(a, b):  
    result = a + b  
    return result
```

```
x = add_numbers(3, 4)  
print(x)
```

script4.py

```
def check_value(value: int):  
    if value > 10:  
        return "Greater than 10"  
    return "Less or equal to 10"  
    print("This is unreachable")
```

```
print(check_value(5))
```

script5.py

```
def append_to_list(item, my_list=[]):  
    my_list.append(item)  
    return my_list
```

```
print	append_to_list(1)  
print	append_to_list(2)
```

$$\textcircled{A} \quad \text{Precision} = \frac{TP}{TP+FP} \quad (100\% \text{ Precision is Soundness})$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (100\% \text{ Recall is Completeness})$$

$FN \leftrightarrow$ Complete
 $FP \leftrightarrow$ Sound

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Lattice :-

Total order is a binary relation \leq (strict total order is $<$)
 linear order set (lorder) is a set equipped with a total order.

which of them are lorders?

$\{-1, 5, 2, 0, 4, 2\} \leftarrow \text{Yes}$

$\{3, 5, -9, 5, 12\} \leftarrow \text{No}$

$\{3, 5, "Hello", 12, 5.0\} \leftarrow \text{No}$

$\{x, y, z\} \leftarrow \text{maybe}$

Partially ordered set

\rightarrow Partially order is total order between some elements

\rightarrow Partially ordered set (poset) is a set equipped with a partial order.

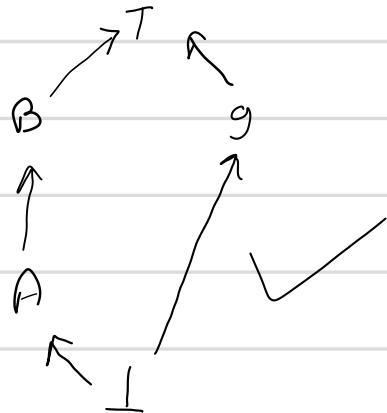
$\{1, "apple", 2, -7, "orange"\} \leftarrow \text{Yes}$



Lattice :- It is a poset where two elements (x, y) have least upper bound (join operator $x \sqcup y$) and greatest lower bound (meet operator $x \sqcap y$)

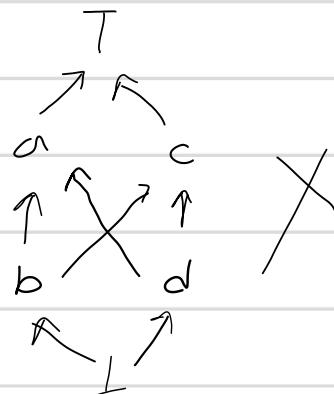
$$\{A, T, g, B, \perp\}$$

$$\{T, \perp, a, b, c, d\}$$



$$A \sqcup g = T$$

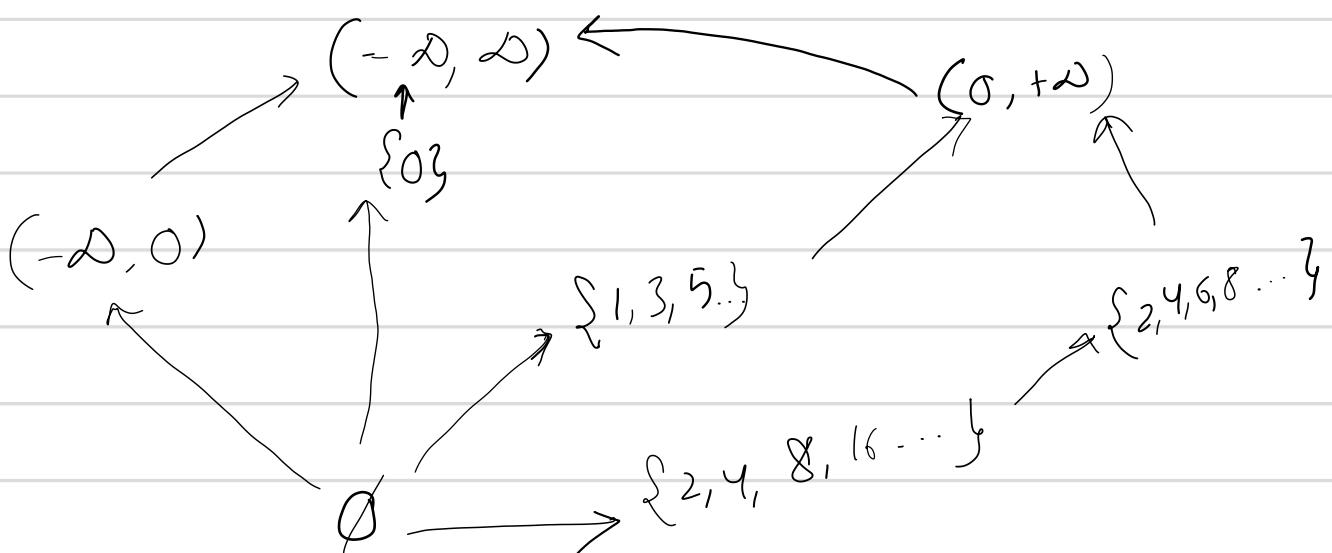
$$B \sqcap g = \perp$$



$$b \sqcup d = \{a, c\}$$

$$a \sqcap c = \{b, d\}$$

Partial order is \in

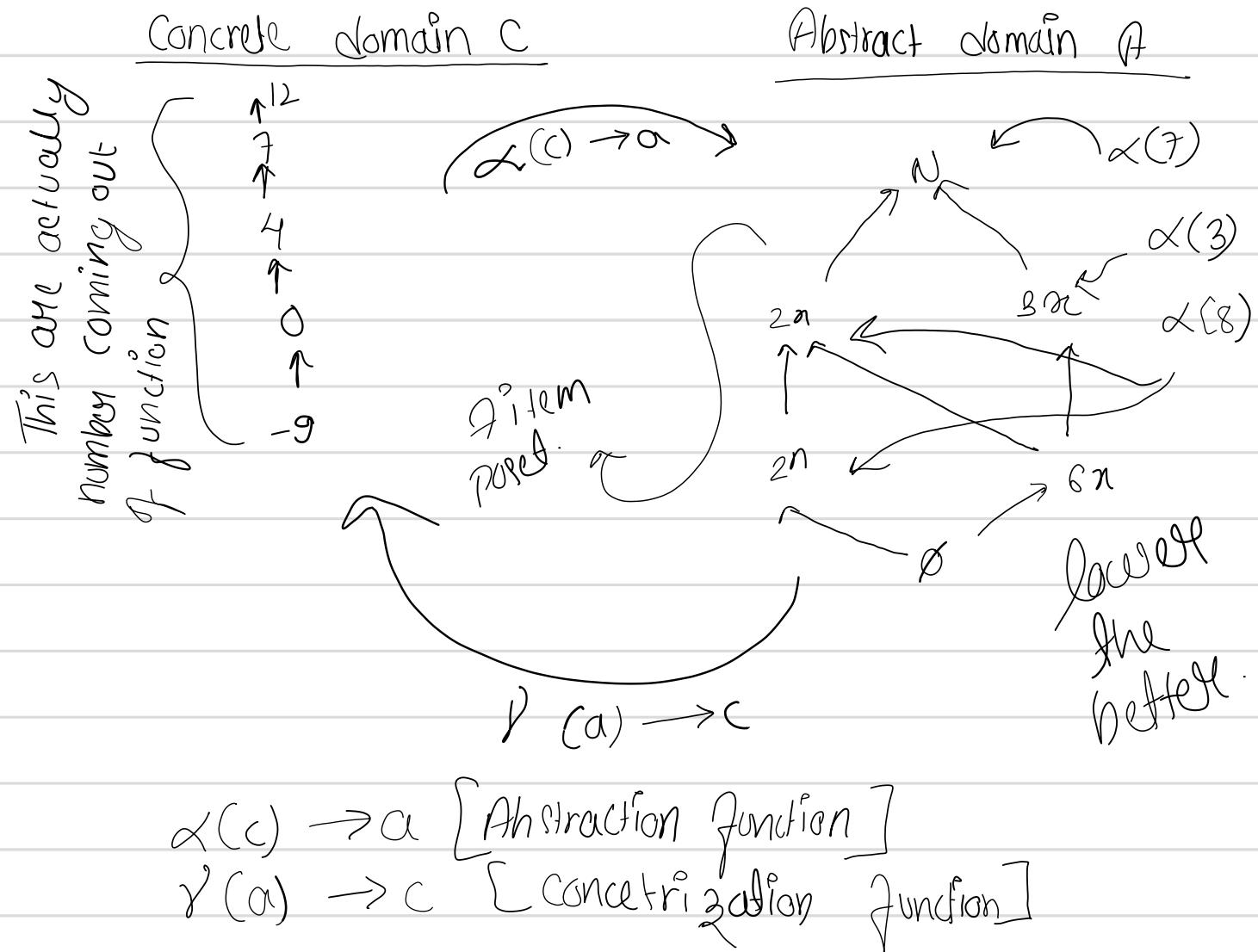


Abstract Interpretation :-

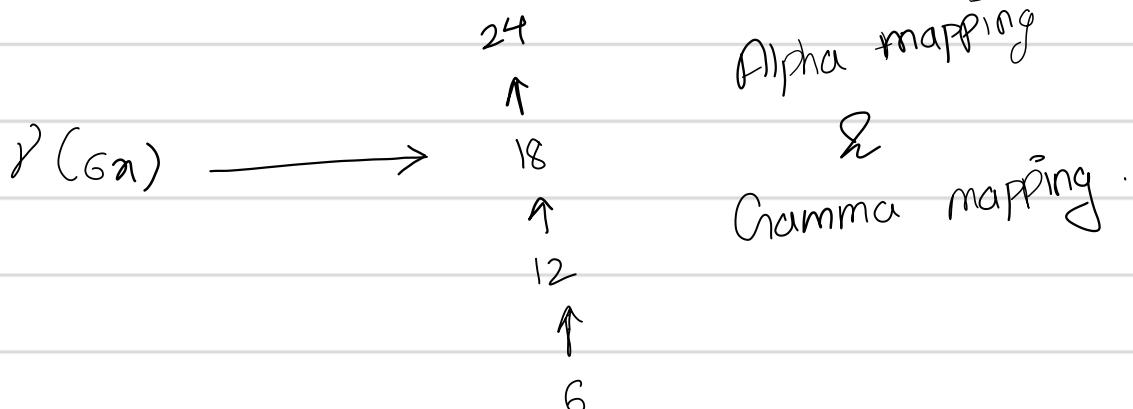
There is a compromise to be made between the precision of the analysis and its decidability (computability) or tractability (computation cost)

Soln is Over & Under approximation.
 " too much " too little.

Abstraction & concretization:



So, lower we put on the lattice, the more precise will be the analysis.



⑦ 1. Ordered sets (Domains)

- ↳ Concrete Domains (C) \rightarrow Precise, detail
- ↳ Abstract Domains (A) \rightarrow approximation
 - \rightarrow Sets of concrete states.

2. Order Relations

Both C and A are poset.

\leq_C for C .

\leq_A for A .

3. Galois connection

A Galois connection between C and A is defined by two function

① Abstraction function (α)

② Concretization function (β)

$$\alpha : C \longrightarrow A$$

$$\beta : A \longrightarrow C$$

This function must satisfy following condition for all $c \in C$ and $a \in A$:

$$\alpha(c) \leq_A a$$

$$\text{iff } c \leq_C \beta(a)$$

This means α and β form a pair of adjoint function, ensuring that the abstraction & concretization are consistent w.r.t the ordering in each domain.

Example

$$[a_1, b_1] \leq_A [a_2, b_2]$$

$$\Rightarrow [10, 20] \leq_A [5, 30]$$



Adjoint pair condition

$$\alpha(c) \leq_A a \iff c \leq_C \beta(a)$$

let, $c = 5$ be a concrete value.

$$\bullet \alpha(5) = [5, 5]$$

let, $a = [4, 6]$ be an abstract interval.

To verify :-

$$\textcircled{1} \quad \alpha(5) \leq_A [4, 6] \iff \textcircled{2} \quad 5 \leq_C \beta([4, 6])$$

check ①

$$[5, 5] \leq_A [4, 6]$$

this is as $4 \leq 5 \leq 6$

check ②

$$5 \leq_C \beta([4, 6])$$

$$5 \leq_C \{z \in \mathbb{Z} \mid 4 \leq z \leq 6\}$$

$$5 \leq_C \{4, 5, 6\}$$

this is also true.

This satisfies Galois connection.



Fixed point calculation :-

FP computation is a repeated symbolic execution of the program using abstract semantics until approximation reaches to an equilibrium.

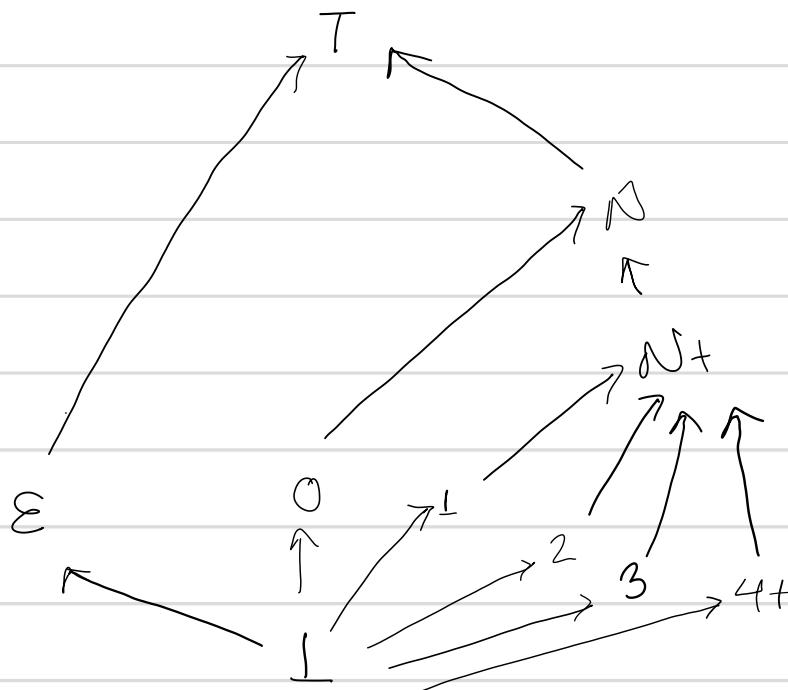
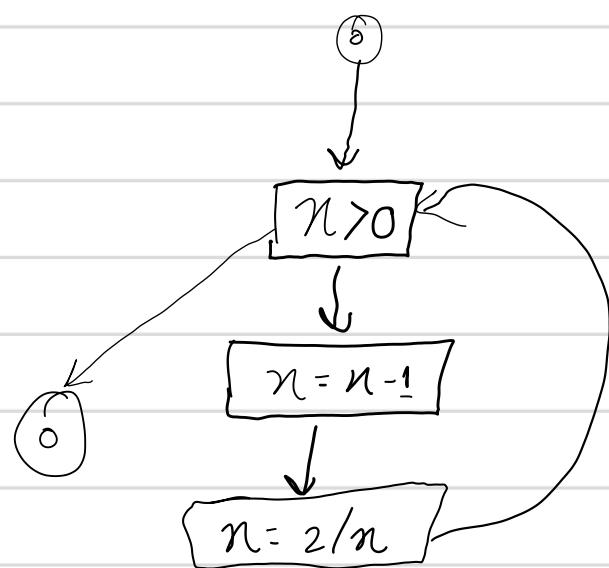
def calc(n):

 while $n > 0$:

$n = n - 1$

$n = 2/n$

 return n



Operational Semantics [meaning to programs]

- Operational Semantics of imperative languages.

~~For Operational semantics.~~

Consider a C code :-

int i = 5;

func(i++, --i);

Show in

chatgpt.

What are the arguments passed to fun?

1. 5, 5 (left to right)
2. 4, 4 (right to left)

Both options are possible in C!

→ Unspecified Semantics.

→ Compiler decides.

want : all behaviour should be clearly specified.

Specifying the Semantics of programs

- Static Semantics eg. type systems.
- Dynamic Semantics
 - ↳ Operational Semantics
- is useful for language design.
- is useful for language implementation
- Programming.
- Program analysis.

a) Transition System

- It describes the state of Systems
- Transition from one state to another.

- Set called config is set of configurations/states
- binary relation (\rightarrow) $\subseteq \text{Config} \times \text{Config}$

"Transition relation"

$a \rightarrow a'$ (transition)

↳ Step of computation

↳ Should be deterministic.

Notation \rightarrow^* (reflexive, transitive closure of \rightarrow)

↓

$\forall a. a \rightarrow^* a$

$\forall a, a', a''$

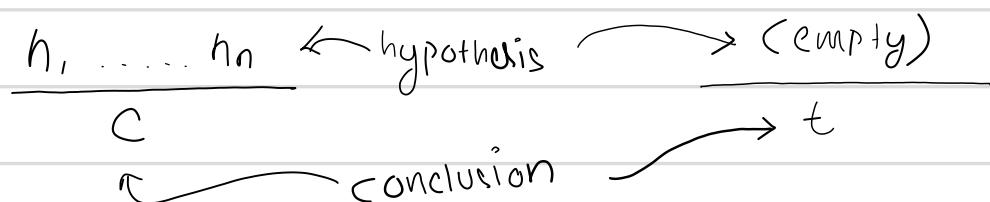
$c \xrightarrow{*} c' \wedge c' \xrightarrow{*} c''$

$c \xrightarrow{*} c''$

b) Rule induction

↳ define a set ("inductive set") with

- ① a finite set of basic elements ("axioms")
- ② a finite set of rules that specify how to generate more elements



Example : Set of natural numbers .

Axiom :

$$\overline{0}$$

Rule :

$$\frac{n}{n+1}$$



Example: Arithmetic expression in a P.L.

$$+(3, 4)$$

↳ Set = pairs of ASI & values after evaluations.

Notation : $E \Downarrow n$; expression E evaluated to number n.

Axioms : $1 \Downarrow 1, 2 \Downarrow 2, \text{etc}$

$\underbrace{\quad\quad\quad}_{\text{Axiom Scheme}} n \Downarrow n$

Rules :

$$\frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{+ (E_1, E_2) \Downarrow n} \quad \text{if } n = n_1 + n_2 \text{ etc.}$$

$\underbrace{\quad\quad\quad}_{\text{rule scheme}}$

rule scheme :

$$\frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{OP(E_1, E_2) \Downarrow n} \quad \text{if } n = n_1 \text{ OP } n_2$$

c) Proof tree

↳ Shows that an element is in an inductive set.

Ex

2 is a natural number

$$\begin{array}{c} \overline{0} \leftarrow \text{It is a axiom} \\ \hline \overline{1} \\ \hline \overline{2} \end{array}$$



Ex 2

Show that

$$-(+(3,4),1) \Downarrow 6$$

$$\begin{array}{r}
 \overline{3 \Downarrow 3} \quad \overline{4 \Downarrow 4} \\
 \hline
 + (3,4) \Downarrow 7 \quad \overline{1 \Downarrow 1} \\
 \hline
 -(+(3,4),1) \Downarrow 6
 \end{array}$$

Axioms

8

2 Rules

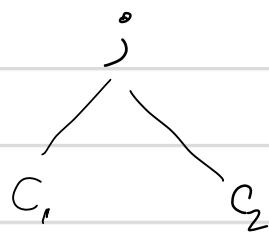
Abstract Syntax of SIMP (Simple Imperative Programming Language)

- Features of SIMP
 - ↳ assignment, integer variables
 - ↳ Sequencing
 - ↳ conditionals
 - ↳ loops

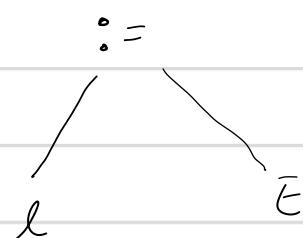
Abstract Syntax

$$P ::= C \mid E \mid B$$

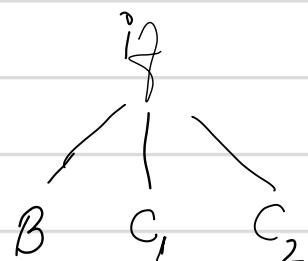
d) commands



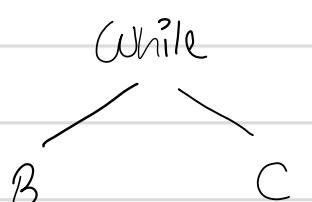
(Sequence)



(Assignment of an expression E to some label l)



(two-way selection)



(while loop)

Skip

(do nothing)



Created with
Notewise

Textual notation :-

$C ::= C; C \mid l ::= E \mid \text{if } B \text{ then } C_1 \text{ else } C_2$

while $B \mid \text{do } C \mid \text{skip}$

consists of

b) Integer Expressions.

$E ::= !l \mid n \mid E \text{ op } E$

$\text{op} ::= + / - / * / /$

n is integer

$l \in C = \{ l_0, l_1, l_2, \dots \}$ memory locations

$!l$ value stored at location l .

c) Boolean Expression

$B ::= \text{True} / \text{False} / E \text{ bop } E / \neg B / B \wedge B$

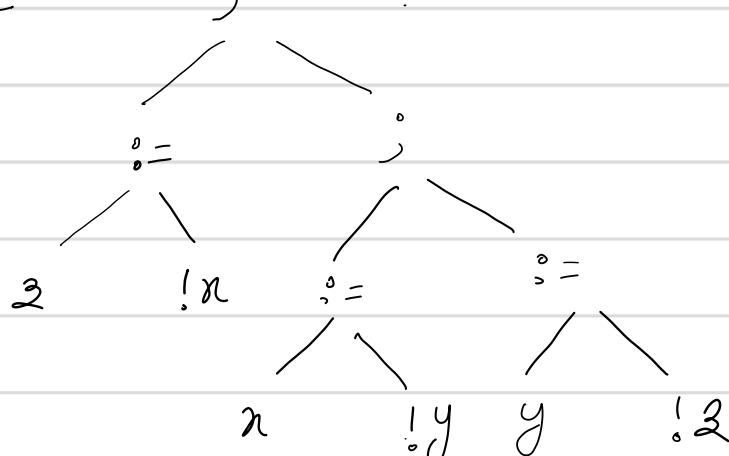
$\text{bop} ::= > / < / =$

Example :-

$z := !n ; (n := !y ; y := !z)$

Swapping values of n and y using z .

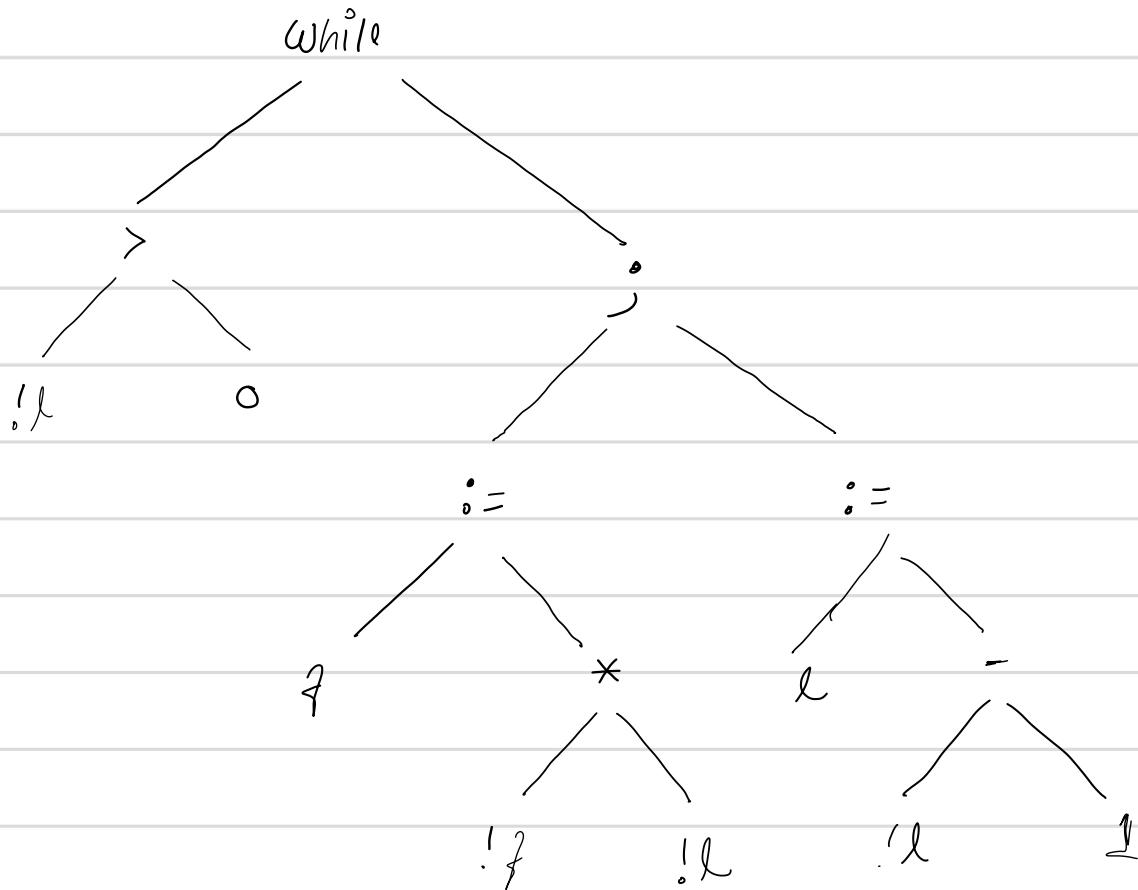
AST :-



Ex 2.

while ($!l > 0$) do (

$f := f * !l;$
 $l := l - 1$)



An abstract machine for SIMP

A hypothetical machine with 4 elements

- (I) Control stack c : Store instruction to execute
- (II) auxiliary stack r : store intermediate results
- (III) processor \circ : performs arithmetic operation, comparison, boolean operations
- (IV) memory stack m : partial function mapping location to value (integer)



Notation : $m[l \mapsto n]$ update function m with new mapping $l \mapsto n$

$$m[l \mapsto n](l) = n$$

$$m[l \mapsto n](l') = m(l') \quad \text{if } l' \neq l$$

Abstract machine

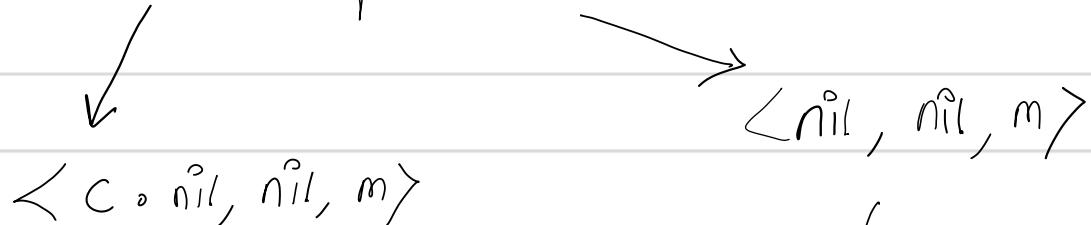
Configuration : $\langle C, r, m \rangle$

$C := \text{nil} / i \circ c$ instruction i push
empty stack on top of stack

$i := p(\text{op}) / \gamma / n / bop / \circ = (\gamma / \text{while})$

$r := \text{nil} / \text{par} / \text{lor}$

Model execution of programs as sequence of transitions from initial state to final state.



Stop when all stacks are empty.

Transition rules :- \rightarrow

$$\langle C, r, m \rangle \rightarrow \langle C', r', m' \rangle$$



Created with
Notewise

a) Evaluating Expressions.

$$\hookrightarrow \langle \text{noc}, r, m \rangle \rightarrow \langle c, \text{nor}, m \rangle$$

poping n from
c \times

push it onto r.

$$\hookrightarrow \langle !\text{loc}, r, m \rangle \rightarrow \langle c, \text{nor}, m \rangle \text{ if } m(l) = 1$$

read the memory at l and push

it onto r.

$$\hookrightarrow \langle (E_1 \circ E_2) \text{oc}, r, m \rangle \rightarrow \langle E_1 \circ E_2 \circ \text{opoc}, r, m \rangle$$

$$\langle \text{op}, n_2 \text{on}, \text{or}, m \rangle \rightarrow \langle c, \text{nor}, m \rangle \text{ if } n_1, \text{op}$$

why this is
opposite.

Semantics of SIMP Expressions.

Value of E in state m is v if there is a seqn of transitions

$$\langle E \text{oc}, r, m \rangle \xrightarrow{*} \langle c, \text{vor}, m' \rangle$$

$m' = m$.

⑧ in case of att $[m' \neq m]$

for SIMP



Example: What is the value of
 $E = !a + !b$ in state
 $m = \{ a \mapsto 3, b \mapsto 1 \}$

Soln.

= $\langle !a + !b \circ \text{nil}, \text{nil}, m \rangle$
 $\rightarrow \langle !a \circ !b \circ \text{nil}, \text{nil}, m \rangle$
 $\rightarrow \langle !b \circ \text{nil}, 3 \circ \text{nil}, m \rangle$
 $\rightarrow \langle \text{nil}, 1 \circ 3 \circ \text{nil}, m \rangle$
 $\rightarrow \langle \text{nil}, 4 \circ \text{nil}, m \rangle$
↑
Ans

b) Executing Commands

$\langle \text{skip} \circ c, r, m \rangle \rightarrow \langle c, r, m \rangle$
$\langle (l := E) \circ c, r, m \rangle \rightarrow \langle E \circ l := c, l \circ r, m \rangle$
push l on auxiliary stack.

$\langle l := c, \text{nil} \circ r, m \rangle \rightarrow \langle c, r, m[l \mapsto n] \rangle$

write n to location l.

$\langle (c_1 ; c_2) \circ c, r, m \rangle \rightarrow \langle c_1 \circ c_2 \circ c, r, m \rangle$
$\langle \text{if } B \text{ then } C_1 \text{ else } C_2 \circ c, r, m \rangle \rightarrow$
 $\langle B \circ \text{if } c, c_1 \circ c_2 \circ r, m \rangle$
↑ Store commands for later.

$\langle \text{if } c, \text{True} \circ c_1 \circ c_2 \circ r, m \rangle$



$\rightarrow \langle C, \circ C, r, m \rangle$

$\langle \text{if } \circ C, \text{False } \circ C_1 \circ r, m \rangle$

$\rightarrow \langle C_1 \circ C, r, m \rangle$

(#) $\langle (C \text{while } B \text{ do } C) \circ C, r, m \rangle$

$\rightarrow \langle B \circ \text{while } \circ C, B \circ C \circ r, m \rangle$

$\langle \text{while } \circ C, \text{True } \circ B \circ C \circ r, m \rangle$

$\rightarrow \langle C \circ (\text{while } B \text{ do } C) \circ C, r, m \rangle$

$\langle \text{while } \circ C, \text{False } \circ B \circ C \circ r, m \rangle$

$\rightarrow \langle C, r, m \rangle$

Semantics of SIMP commands

Program C execute state m terminates successfully

& produces state m' if there is a sequence of transitions.

$\langle C \circ \text{nil}, \text{nil}, m \rangle \xrightarrow{*} \langle \text{nil}, \text{nil}, m' \rangle$



Example: $C = \text{while } l > 0 \text{ do } \{ f := !f * !l \};$

$$C' = \underbrace{\text{while } l > 0 \text{ do }}_{\substack{l := !l - 1 \\ C_1}} \underbrace{\{ f := !f * !l \}}_{\substack{f := !f \\ C_2}}$$

$$m = \{ l \mapsto 4, f \mapsto 1 \}$$

Soln.

$\langle \text{Conil}, \text{nil}, m \rangle$

$$\rightarrow \langle \text{Bo while o nil}, \text{Bo} C' \circ \text{nil}, m \rangle$$

$$\rightarrow \langle !l \circ \text{Do} > \text{o while o nil}, \text{Bo} C' \circ \text{nil}, m \rangle$$

$$\rightarrow \langle \text{Do} > \text{o while o nil}, 4 \circ \text{Bo} C' \circ \text{nil}, m \rangle$$

$$\rightarrow \langle > \text{o while o nil}, 0040 \text{Bo} C' \circ \text{nil}, m \rangle$$

$$\rightarrow \langle \text{while o nil}, \text{True} \circ \text{Bo} C' \circ \text{nil}, m \rangle$$

$$\rightarrow \langle C' \circ \text{Conil}, \text{nil}, m \rangle$$

$$\rightarrow \langle C_1 \circ C_2 \circ C_0 \text{nil}, \text{nil}, m \rangle$$

$$\rightarrow \langle !f * !l \circ := \circ C_2 \circ C_0 \text{nil}, f \circ \text{nil}, m \rangle$$

$$\rightarrow \langle !f \circ !l \circ * \circ := \circ C_2 \circ C_0 \text{nil}, f \circ \text{nil}, m \rangle$$

$$\rightarrow \langle !l \circ * \circ := \circ C_2 \circ C_0 \text{nil}, 1 \circ f \circ \text{nil}, m \rangle$$

$$\rightarrow \langle * \circ := \circ C_2 \circ C_0 \text{nil}, 4 \circ f \circ \text{nil}, m \rangle$$

$$\rightarrow \langle \circ := \circ C_2 \circ C_0 \text{nil}, 4 \circ f \circ \text{nil}, m \rangle$$

$$\rightarrow \langle C_2 \circ C_0 \text{nil}, \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle !l - 1 \circ := \circ C_0 \text{nil}, l \circ \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle !l \circ 1 \circ - \circ := \circ C_0 \text{nil}, l \circ \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle 1 \circ - \circ := \circ C_0 \text{nil}, 4 \circ l \circ \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle - \circ := \circ C_0 \text{nil}, 1 \circ 4 \circ l \circ \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle := \circ C_0 \text{nil}, 3 \circ l \circ \text{nil}, m \{ f \mapsto 4 \} \rangle$$

$$\rightarrow \langle C_0 \text{nil}, \text{nil}, m \{ f \mapsto 4, l \mapsto 3 \} \rangle$$

$\frac{\circ}{\circ}$

$\rightarrow \langle \text{nil}, \text{nil}, m \{ f \mapsto 4, l \mapsto 0 \} \rangle$



1. Evaluation of Expressions:

$\langle n \cdot c, r, m \rangle$	\rightarrow	$\langle c, n \cdot r, m \rangle$	
$\langle b \cdot c, r, m \rangle$	\rightarrow	$\langle c, b \cdot r, m \rangle$	
$\langle \neg B \cdot c, r, m \rangle$	\rightarrow	$\langle B \cdot \neg \cdot c, r, m \rangle$	
$\langle (B_1 \wedge B_2) \cdot c, r, m \rangle$	\rightarrow	$\langle B_1 \cdot B_2 \cdot \wedge \cdot c, r, m \rangle$	
$\langle \neg \cdot c, b \cdot r, m \rangle$	\rightarrow	$\langle c, b' \cdot r, m \rangle$	if $b' = \text{not } b$
$\langle \wedge \cdot c, b_2 \cdot b_1 \cdot r, m \rangle$	\rightarrow	$\langle c, b \cdot r, m \rangle$	if $b_1 \text{ and } b_2 = b$
$\langle (E_1 \text{ op } E_2) \cdot c, r, m \rangle$	\rightarrow	$\langle E_1 \cdot E_2 \cdot \text{op} \cdot c, r, m \rangle$	
$\langle (E_1 \text{ bop } E_2) \cdot c, r, m \rangle$	\rightarrow	$\langle E_1 \cdot E_2 \cdot \text{bop} \cdot c, r, m \rangle$	
$\langle \text{op} \cdot c, n_2 \cdot n_1 \cdot r, m \rangle$	\rightarrow	$\langle c, n \cdot r, m \rangle$	if $n_1 \text{ op } n_2 = n$
$\langle \text{bop} \cdot c, n_2 \cdot n_1 \cdot r, m \rangle$	\rightarrow	$\langle c, b \cdot r, m \rangle$	if $n_1 \text{ bop } n_2 = b$
$\langle !l \cdot c, r, m \rangle$	\rightarrow	$\langle c, n \cdot r, m \rangle$	if $m(l) = n$

2. Evaluation of Commands:

$\langle \text{skip} \cdot c, r, m \rangle$	\rightarrow	$\langle c, r, m \rangle$	
$\langle (l := E) \cdot c, r, m \rangle$	\rightarrow	$\langle E \cdot := \cdot c, l \cdot r, m \rangle$	
$\langle := \cdot c, n \cdot l \cdot r, m \rangle$	\rightarrow	$\langle c, r, m[l \mapsto n] \rangle$	
$\langle (C_1; C_2) \cdot c, r, m \rangle$	\rightarrow	$\langle C_1 \cdot C_2 \cdot c, r, m \rangle$	
$\langle (\text{if } B \text{ then } C_1 \text{ else } C_2) \cdot c, r, m \rangle$	\rightarrow	$\langle B \cdot \text{if} \cdot c, C_1 \cdot C_2 \cdot r, m \rangle$	
$\langle \text{if} \cdot c, \text{True} \cdot C_1 \cdot C_2 \cdot r, m \rangle$	\rightarrow	$\langle C_1 \cdot c, r, m \rangle$	
$\langle \text{if} \cdot c, \text{False} \cdot C_1 \cdot C_2 \cdot r, m \rangle$	\rightarrow	$\langle C_2 \cdot c, r, m \rangle$	
$\langle (\text{while } B \text{ do } C) \cdot c, r, m \rangle$	\rightarrow	$\langle B \cdot \text{while} \cdot c, B \cdot C \cdot r, m \rangle$	
$\langle \text{while} \cdot c, \text{True} \cdot B \cdot C \cdot r, m \rangle$	\rightarrow	$\langle C \cdot (\text{while } B \text{ do } C) \cdot c, r, m \rangle$	
$\langle \text{while} \cdot c, \text{False} \cdot B \cdot C \cdot r, m \rangle$	\rightarrow	$\langle c, r, m \rangle$	

4.1 Transition rules of the abstract machine for SIMP

Structural O.S. $\xrightarrow{\quad}$ Small - Step Semantics
 $\xrightarrow{\quad}$ Big - Step Semantics ..

Small - Step Semantics / Reduction Semantics

- Transition system with
 - configuration $\langle P, S \rangle$ P is program
 S is store —
(function from locations to integers)
- Transition relation between configurations. Is defined via axioms and rules.

Axioms & rules for expressions.

$$\frac{}{\langle !l, S \rangle \rightarrow \langle n, S \rangle} \text{ if } S(l) = n \quad (\text{var})$$

$$\frac{}{\langle n_1, op\ n_2, S \rangle \rightarrow \langle n, S \rangle} \text{ if } n = (n_1, op, n_2) \quad (\text{op})$$

$$\frac{\langle E_1, S \rangle \rightarrow \langle E'_1, S' \rangle}{\langle E, op\ E_2, S \rangle \rightarrow \langle E'_1 \ op\ E'_2, S' \rangle} \quad (\text{op}_c)$$

$$\frac{\langle E_2, S \rangle \rightarrow \langle E'_2, S' \rangle}{\langle n, op\ E_2, S \rangle \rightarrow \langle n, op\ E'_2, S' \rangle} \quad (\text{op}_r)$$

Axioms for commands

$$\textcircled{1} \quad \frac{}{\langle l := n, s \rangle \rightarrow \langle \text{skip}, s[l \mapsto n] \rangle} (=)$$

$$\textcircled{2} \quad \frac{\langle E, s \rangle \rightarrow \langle E', s' \rangle}{\langle l := E, s \rangle \rightarrow \langle l := E', s' \rangle} (=)_R$$

$$\textcircled{3} \quad \frac{}{\langle \text{skip} ; c, s \rangle \rightarrow \langle c, s \rangle} (\text{skip})$$

$$\textcircled{4} \quad \frac{\langle c_1, s \rangle \rightarrow \langle c'_1, s' \rangle}{\langle c_1 ; c_2, s \rangle \rightarrow \langle c'_1 ; c_2, s' \rangle} (\text{sequence})$$

$$\textcircled{5} \quad \frac{}{\langle \text{if True then } c_1 \text{ else } c_2, s \rangle \rightarrow \langle c_1, s \rangle} (\text{if}_T)$$

$$\textcircled{6} \quad \frac{}{\langle \text{if False then } c_1 \text{ else } c_2, s \rangle \rightarrow \langle c_2, s \rangle} (\text{if}_F)$$

$$\textcircled{7} \quad \frac{\langle B, s \rangle \rightarrow \langle B', s' \rangle}{\langle \text{if } B \text{ then } c_1 \text{ else } c_2, s \rangle \rightarrow \langle \text{if } B' \text{ then } c_1 \text{ else } c_2, s' \rangle} (\text{if_rule})$$

$$\textcircled{8} \quad \frac{}{\langle \text{while } B \text{ do } c, s \rangle \rightarrow \langle \text{if } B \text{ then } (c ; \text{while } B \text{ do } c) \text{ else skip}, s \rangle} (\text{while})$$

Ex:

$$P_0 \quad z := !n ; (n := !y ; y := !z)$$

$$S = \{ z \mapsto 0, n \mapsto 1, y \mapsto 3 \}$$

Except of proof tree:-

$$\langle z := !n ; (n := !y ; y := !z), S \rangle$$

$$\rightarrow \langle z := 1 ; (n := !y ; y := !z), S \rangle$$

$$\rightarrow \langle \text{Skip} ; (n := !y ; y := !z), S[z \mapsto 1] \rangle$$

$$\rightarrow \langle n := !y ; y := !z, S[z \mapsto 1] \rangle$$

$$\rightarrow \langle \text{Skip} ; y := !z, S[z \mapsto 1, n \mapsto 2] \rangle$$

$$\rightarrow \langle y := !z, S[z \mapsto 1, n \mapsto 2] \rangle$$

$$\rightarrow \langle \text{Skip}, S[z \mapsto 1, n \mapsto 2, y \mapsto 1] \rangle$$

N.

Big - Step Semantics

$$\langle P, S \rangle \Downarrow \langle P^1, S' \rangle$$

evaluates
 to

evaluation
 relation

$$\overline{\langle !l, S \rangle \Downarrow \langle n, S \rangle \text{ if } S(l) = n} \quad (\forall l)$$

axioms

$\langle \bar{e}, s \rangle \Downarrow \langle n, s' \rangle$ $\underbrace{\langle l_0^0 = E, s \rangle \Downarrow \langle \text{skip}, s' [l \rightarrow n] \rangle}_{:=}$ ~~\mathcal{E}^N~~ $(z^0 = !n ; n^0 = !y) ; y^0 = !z$ $s(z) = 0, s(n) = 1, s(y) = 2.$

Prove :- $\langle p, s \rangle \Downarrow \langle \text{skip}, s' \rangle$ using
~~Big~~ Evaluation Relation.

 $\langle z^0 = n, s \rangle \Downarrow \langle \text{skip}, s[z \mapsto !] \rangle :$ $\overbrace{\quad\quad\quad}^{(\text{var})} \langle !n, s \rangle \Downarrow \langle !, s \rangle$ (\because) $\langle z^0 = !n, s \rangle \Downarrow \langle \text{skip}, s[z \mapsto !] \rangle$ again, $\overbrace{\quad\quad\quad}^{(\text{var})}$ $\langle !y, s[y \mapsto !] \rangle \Downarrow \langle 2, [z \mapsto !] \rangle$ $\underbrace{\langle n^0 = !y, s[z \mapsto !] \rangle \Downarrow \langle \text{skip}, s[z \mapsto !, n \mapsto 2] \rangle}_{(\because)}$

and so on book page no 83.

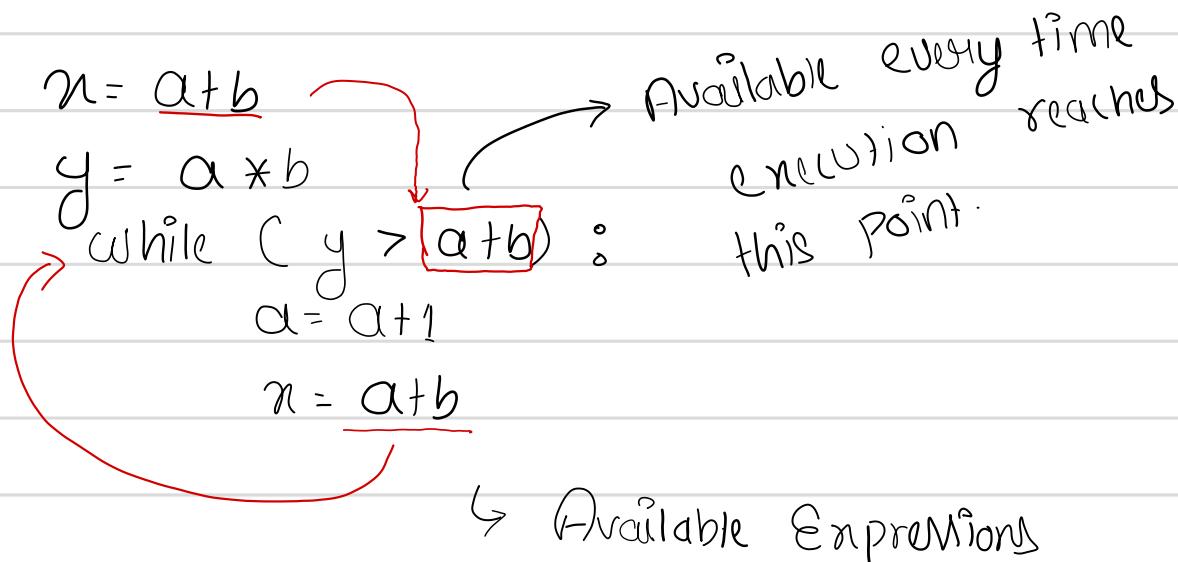


Exercise 1 : Operational Semantics :-

Data flow Analysis \Rightarrow Static Analysis

Available Expression analysis

- ↳ For each program point (code, Statement), compute which expressions must have already been computed, & not later modified.
- ↳ avoids re-computing
- ↳ compiler optimizations.



Transfer Functions

- ↳ How the Statement affects the analysis state.
- ↳ available expressions.

Two functions

- gen :- available expressions generated.
- Kill :- "

① gen function :-

gen : Stmt \rightarrow P(Expression)

Ex :-

$n = a * b$ generates $a * b$

② Kill function

Kill : Stmt \rightarrow P(Expression)

example

$a = 23$

Kill

$a * b$

CFG

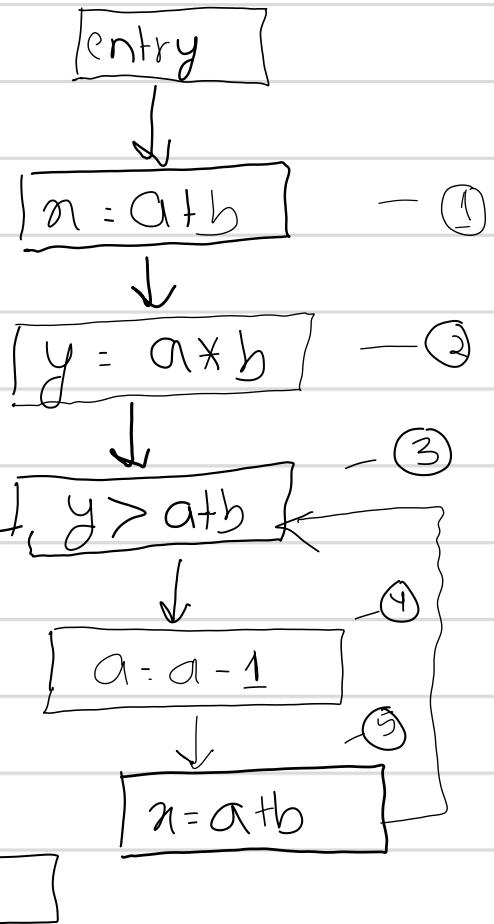
$n = a + b$

$y = a * b$

while $y > a + b$:

$a = a + 1$

$n = a + b$



Non-trivial Expression

$a+b$

$a \times b$

$a - 1$

Transfer function for each Stmt:

Stmt	gen(s)	kill(s)
1.	{ $a+b$ }	∅
2	{ $a \times b$ }	∅
3	{ $a+b$ }	∅
4	∅	{ $a+b, a \times b, a-1$ }
5	$a+b$	∅

Propagating Available Expression

↳ Forward analysis.

↳ for each statement S, outgoing available expressions are :-

incoming avail. expression. minus kill(S) plus gen(S).

Data flow equations :-

$AE_{entry}(S) \rightarrow$ available expression at entry of S.

$AE_{exit}(S) \rightarrow$ " " " exit of S.

$$AE_{entry}(1) = \emptyset$$

$$AE_{entry}(2) = AE_{exit}(1)$$



$$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$$
$$AE_{entry}(4) = AE_{exit}(3)$$
$$AE_{entry}(5) = AE_{exit}(4)$$
$$AE_{exit}(1) = AE_{entry}(1) \cup \{atb\}$$
$$AE_{exit}(2) = AE_{entry}(2) \cup \{axb\}$$
$$AE_{exit}(3) = AE_{entry}(3) \cup \{atb\}$$
$$AE_{exit}(4) = AE_{entry}(4) \setminus \{atb, axb, a^{-1}\}$$
$$AE_{exit}(5) = AE_{entry}(5) \cup \{atb\}$$

S	AE _{entry} (S)	AE _{exit} (S)
1	\emptyset	$\{atb\}$
2	$\{atb\}$	$\{atb, axb\}$
3	$\{atb\}$	$\{atb\}$
4	$\{atb\}$	\emptyset
5	\emptyset	$\{atb\}$

Q.

$$m = n - y$$

if (random()) :

while $m > 0$:

$$n = y + 1$$

else :

$$n = n - y$$

$$z = n - y$$

$$\in n - y$$

an available expression

when

decreasing

this
statements
??

Basic principles :-

- (1) Domain
- (2) Direction
- (3) Transfer function
- (4) Meet operator
- (5) Boundary condition
- (6) Initial values.

forward

backward

→ All possible elements the set may have available expr → Non-trivial expression.
→ how a statement affects the propagated information

$$DF_{exit}(s) = \text{Some function of } DF_{entry}(s)$$

Meet operator :-

↳ two statements s_1 and s_2 flow to a statement s .

$$\text{Union: } DF_{entry}(s) = DF_{exit}(s_1) \cup DF_{exit}(s_2)$$

$$\text{Intersection: } DF_{entry}(s) = DF_{exit}(s_1) \cap DF_{exit}(s_2)$$



Boundary Condition :-

Information to start with at the first CFG node.

Initial values :- Information to start with at intermediate nodes.

$$AE_{exit}(s) = (AE_{entry} \setminus kill(s)) \cup gen(s)$$

Defining a Data Flow Analysis

Any data flow analysis:

Defined by six properties

- | | |
|----------------------|--------------------------------------------------------------------|
| ■ Domain | ■ Non-trivial expressions |
| ■ Direction | ■ Forward |
| ■ Transfer function | ■ $AE_{exit}(s) =$
$(AE_{entry} \setminus kill(s)) \cup gen(s)$ |
| ■ Meet operator | ■ Intersection (\cap) |
| ■ Boundary condition | ■ $AE_{entry}(entryNode) = \emptyset$ |
| ■ Initial values | ■ \emptyset |

Example: Available expressions 24 - 2



Examples :-

- ① Reaching Definitions
- ② Very busy Expressions
- ③ Live Variables.

①

Goal :- For each program point, compute which assignments may have been made and may not have been overwritten

$$n = 5$$

$$y = 1$$

while ($x > 1$) :

$$y = n * y$$

$$n = n - 1$$

All definitions
reach the entry
of this statement.

$$n = 5$$

$$y = 1$$

while ($n > 1$) :

$$y = n * y$$

$$n = n - 1$$

6 properties

i) Domain :-

Set of pairs (v, s)

variable

stmt.



3) Direction :- forward.

3) Meet operator :- Union

4) Transfer function :-

$$RD_{exit} = (RD_{entry}(s) \setminus KILL(s)) \cup gen(s)$$

a) $gen(s)$

= If s is assignment to v : (v, s)

b) $KILL(s)$

= If s is assignment to v : (v, s') for
all s' that define v .

5) Boundary condition.

→ Entry node starts with all variables undefined.

$$RD_{entry}(\text{entryNode}) = \{(v, ?) \mid v \in \text{Var}\}$$

6) Initially, all nodes have no reading definitions



$x = 5$

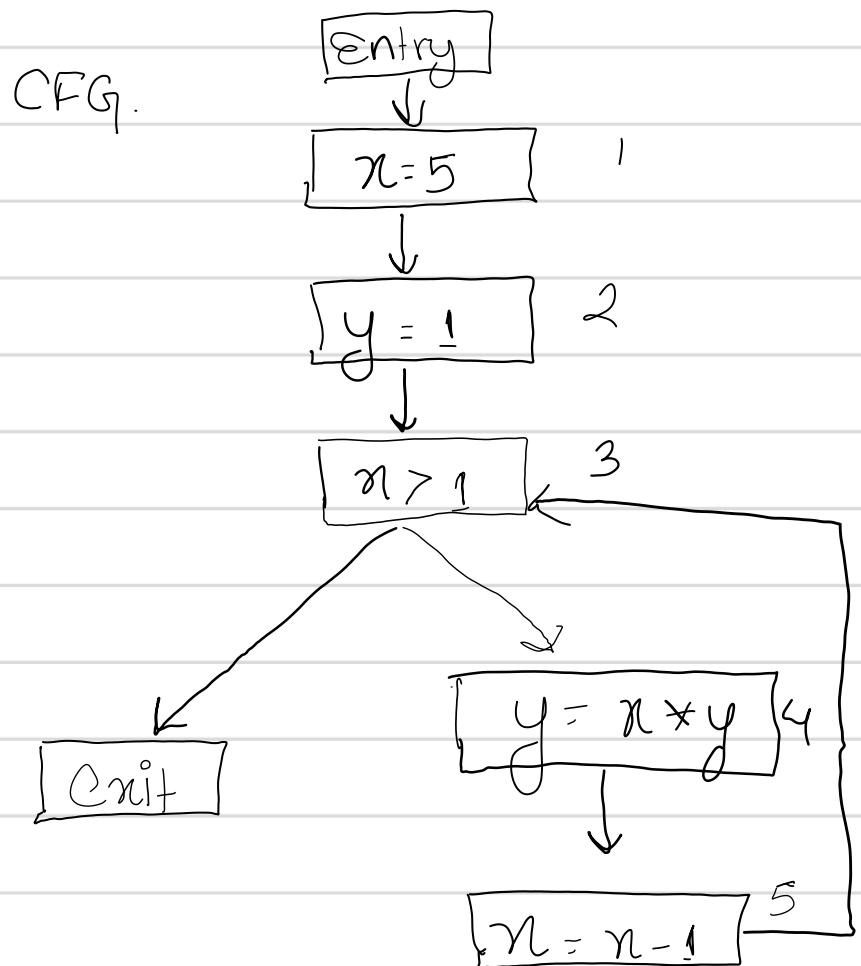
$y = 1$

while $n > 1$:

$y = n * y$

$n = n - 1$

CFG.



\$

gen(s)

1.

{(x, 1)}

2.

{(y, 2)}

3.

\emptyset

4.

{(y, 4)}

5

{(x, 5)}

kill(s)

{(x, 1), (x, 5), (x, ?)}

{(y, 2), (y, 4), (y, ?)}

\emptyset

{(y, 2), (y, 4), (y, ?)}

{(x, 1), (x, 5), (x, ?)}

Data flow equations

$$RDentry(1) = \{(x, ?), (y, ?)\}$$

$$RDentry(2) = RDexit(1)$$

$$RDentry(3) = RDexit(2) \cup RDexit(5)$$

$$RDentry(4) = RDexit(3)$$

$$RDentry(5) = RDexit(4)$$



$$RDexit(1) = (RDentry(1) \setminus \{(n,1), (n,5), (n,?)\}) \cup \{(x,1)\}$$

$$RDexit(2) = (RDentry(2) \setminus \{(y,1), (y,4), (y,?)\}) \cup \{(y,2)\}$$

$$RDexit(3) = RDentry(3)$$

$$RDexit(4) = (RDentry(4) \setminus \{(y,2), (y,4), (y,?)\}) \cup \{(y,4)\}$$

$$RDexit(5) = (RDentry(5) \setminus \{(n,1), (n,5), (x,?)\}) \cup \{(n,5)\}$$

Solution :-

S	<u>RDentry(S)</u>	<u>RDexit(S)</u>
1	$\{(x,?), (y,?)\}$	$\{(x,1), (y,?)\}$
2	$\{(x,1), (y,?)\}$	$\{(x,1), (y,2)\}$
3	$\{(x,1), (y,2), (y,4), (x,5)\}$	$\{(x,1), (y,1), (y,4), (n,5)\}$
4	"	$\{(n,1), (y,4), (n,5)\}$
5	$\{(n,1), (y,4), (n,5)\}$	$\{(y,4), (x,5)\}$

Q. Very Busy Expression Analysis - S.K
 Q. Give Variable " - A.K

(H) Intra - Procedural Analysis.

↳ Reasons about a function in isolation.

Intra - Procedural Analysis:

↳ Reasons about multiple functions.

↳ Calls and returns.

↳ One CFG per function.

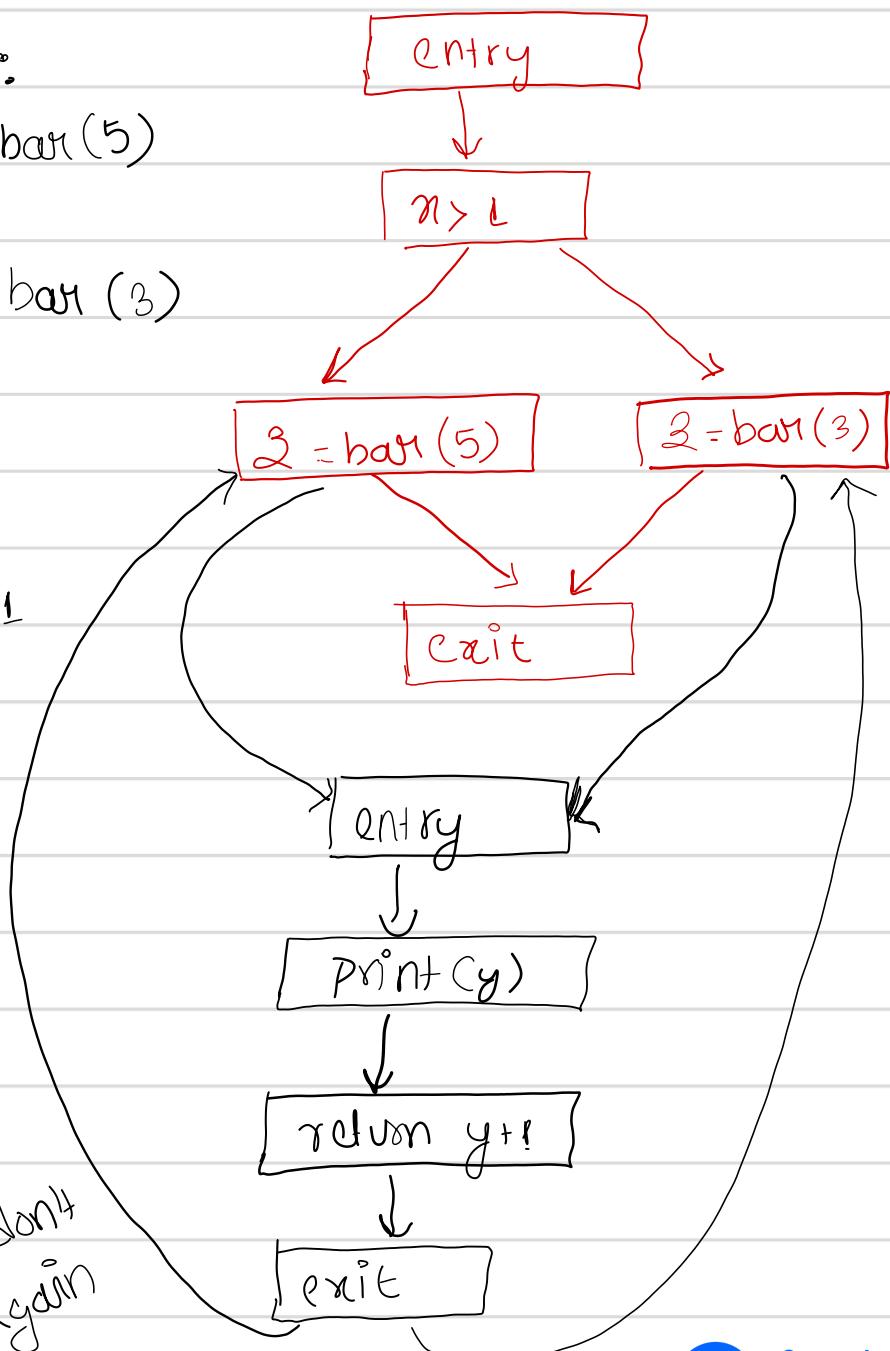
↳ Connect call sites to entry node of callee

↳ Connect exit node back to call site.

Example:-

```
def fun(x):
    if n > 1:
        z = bar(5)
    else:
        z = bar(3)
```

```
def bar(y):
    print(y)
    return y+1
```



When returning, go
back to
call site.

Sensitivity :-

- ① Flow - Sensitive :- Takes into account the order of statements.
- ② Path - Sensitive :- Takes into account the predicates at conditional branches.
- ③ Context - Sensitive :- Takes into account the specific call site that leads into another account.
(inter-procedural analysis only).

Example :-

if ... :

$n = 3$

$n = 5$

↑
Value of n ?

Flow Sensitive = 5

Flow-insensitive = 3 or 5



Path Sensitivity

Path Sensitive: NO
(4)

$$n = 0$$

if $a > 0$:

$$n = \underline{1} \leftarrow$$

else

$$n = 2$$

if $a > 0$:

$$n = n + 3 \leftarrow$$

Can n have value 5:

Path-insensitive :- yes.

Context Sensitivity :-

Context-insensitive :- yes.

$$n = \underline{1}$$

def $f(x)$:

if x :

$$g(3)$$

else:

$$n = 3$$

$$g(5)$$

(congates all call sites of g)

Context-sensitive :-

NO.

def $g(y)$:

can n be equal to

$$y?$$



Call graph Analysis:-

↳ call graph :- Abstraction of all method calls in a program.

- Nodes : Methods
- Edges : calls
- Flow - insensitive.





Created with
Notewise