U19CS012 S.E. BHAGYA RANA

① Metrics - (i) LOC

(Important)

Requirement Analysis & Specification
(1st Phase)

① Feasibility

② Info Gathering

③ Info Analysis

④ Model (DFD, U..)

ambiguity ← • Informal (english) requirements specification

• semi-formal

• formal (maths)

(DFD/ ER Diagrams) Trade Off ↑ Time
(But Accuracy ↑)

Output = [SRS document]

2nd Design & Specification

3rd Coding & ∫ [code & Module ~~testing~~] Implementation

4th Testing Integrating

5th Maintainence (Delivery & Maintence)

~~Evaluation~~

Date
Page

Problem

Student system	Industrial strength s/w
• Built to illustrate something	
• hobby	
• X solves real life problem	• Significant / indirect Impact on Business
- small	- large
- sole user	- Large no. of user
- single	- team of dev's
- lacks detailed req^n	- Detailed req's (desirable)
- Lacks proper user interface	- well designed Interface
- Lacks proper documentation	- well documented & user - manual prepared
- Less efforts in testing	- Rigorous testing
- Ad hoc development (Build & fix) (trial & error)	- systematic development

IEEE defn of s/w — Collection of
- computer programs
- procedures
- rules &
- associated.

s/w
— late & unreliable (due to complexity of s/w)

(35% projects = runaway)

X No aging of software
→ Bugs introduced during the design & development process

* Software Aging → rejuvenation

- memory leak

Software

within time

within budget

Easy to change (dynamic build)

Software are
- Expensive {LOC}
- Late & Unreliable {↑ due to aging}
-

Maintenana & Rework
↳ {Longest}

① Corrective Maintenca

② Adaptive Maintenca

③

$\frac{\text{Maintainee}}{\text{Development}} = \frac{70}{30} \quad \frac{60}{40}$

Scale
of s/w

< 10 KLOC

< 1 MLOC

< 100 KLOC

* (Qualities of software)

→ trade off acc to needs

✓ Functionality
✓ Reliability → defense
✓ Usability → games
✓ Efficiency → embedded system & memory req }
✓ Maintainability
✓ Portability

s/w Lifecycle

S/w Process Models

Jan's Sommerville

Plan-driven Process

- Embedded system (like washing machine)
- Defense (critical system)
- Large & complex system

fixed H/w (worldwide)

Agile

- set of requirement's keep's on changing
- web services
- games

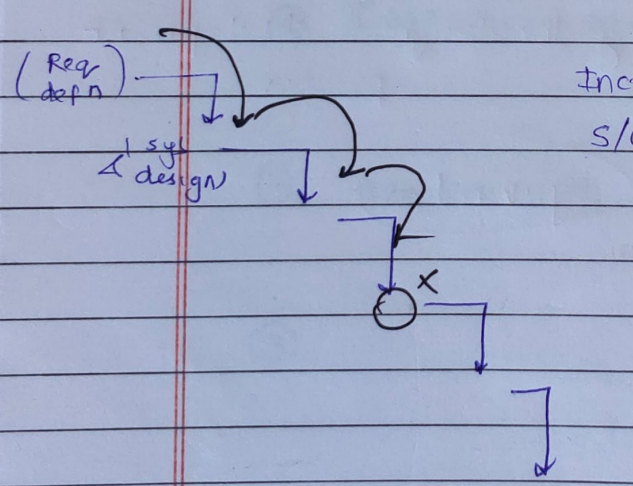
[Generally, Hybrid model is followed]

3 Categories

(plan-driven) → Waterfall

Incremental S/w

Reusability



x no flexibility

difficulty of accomodating change ~~after pro~~

x changing customer requirement

↑ more time in requirement analysis

[More detailed & SRS]

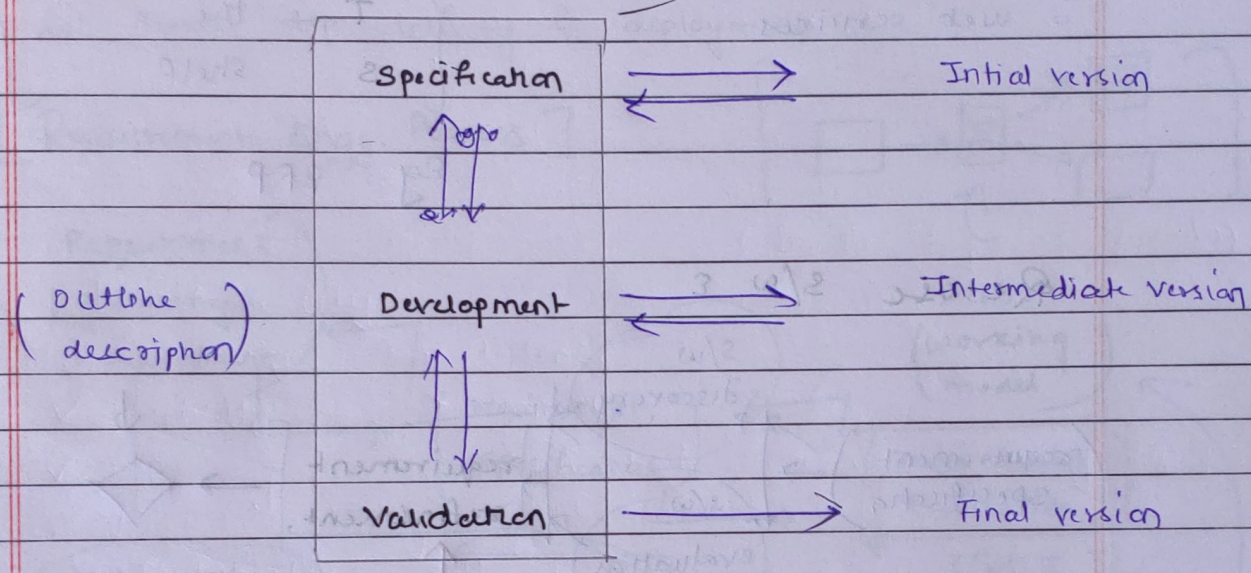
x Customer Feedback.

~~Wednesday~~

2nd Process Model

(B) INCREMENTAL Development

- requirements are freq changing
- less documentation



- customer feedback
- rapid delivery & deployment of S/W

Disadv — ① Process not visible
② system structure degrade

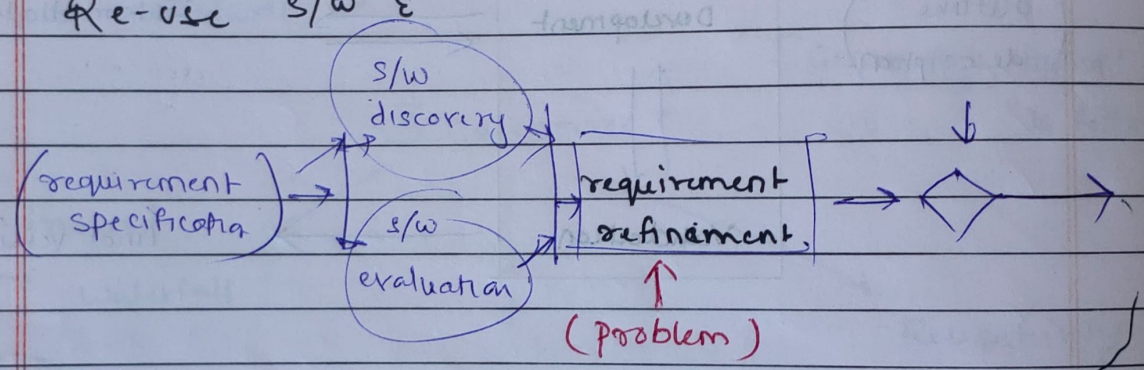
(c) Integration & Configuration

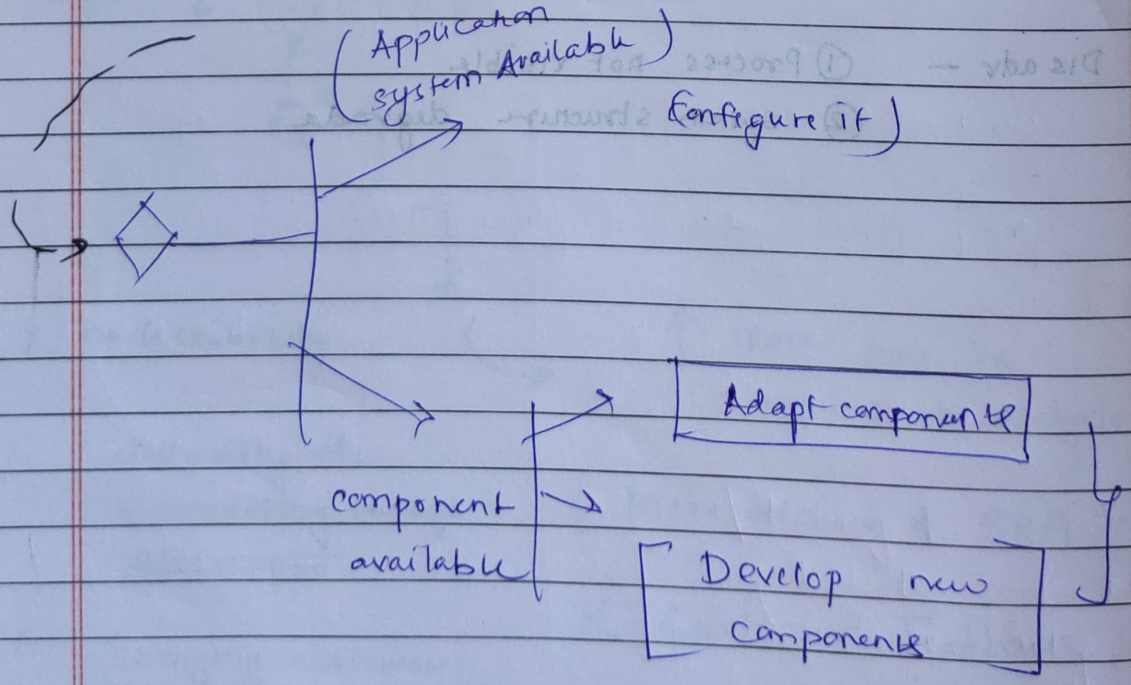
- Reuse Existing Component & libraries
- web services

C O T S — Commercial off the Shelf

Eg: ERP

Re-use s/w ε

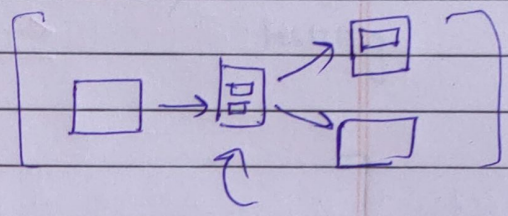




Adv

① ↓ Reduced Cost
② ↓ Risk [∵ less s/w is developed from scratch]
③ Faster ~~dep~~ delivery & deployment

[Requirement Engg. Process]



Requirement gathering & requirement

Demo (working model)

requirement specification

requirement validation

(GUI)

High level system description + User & system requirement

throw away Prototype

+ Data model

SRS

• Validate design during design phase

Interface = Abstraction

stack → × Implementation is hidden

→ push()
pop()
top()
empty()

[Software Prototyping]

Design Process

Architectural Design ⟷ Interface design ⟷ Component design

Database design

Verification

Validation

[Testing]
(test the component for functionality)

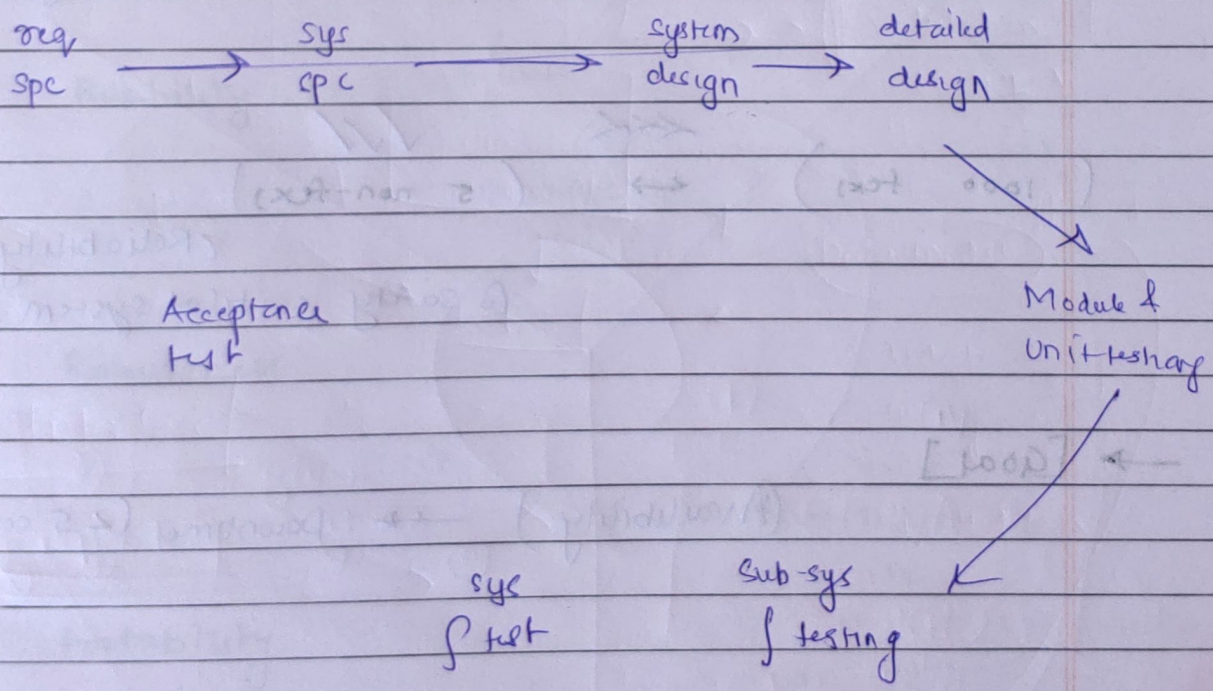
Stages of Testing

{Client}

[Unit Testing] → Component Testing → [Integration testing] (System testing) → Acceptance testing

Famous V model for Testing

(Starts from 1st stage)



Date
Page

(*) types of Non-Fcx Requirements

- product req^r
- organizational req^r
- External req^r

(Govt, Law enforca

① regularility

② Ethical

③ Legislative

(1000 fcx) <<< <-> (5 non-fcx) ✓✓✓

(Reliability)

(o safety critical system)

→ [Goal]

(Availibility) → Downtime (< 5 seconds)

Date
Page

Verifiable Non-f(x) requirements

Speed
- processed transactions / second
- User/
- screen refresh rate

Size
- Mbytes (Embedded system)
- No. of ROM chips

Ease of use
- Training time
- No. of help frames

Reliability

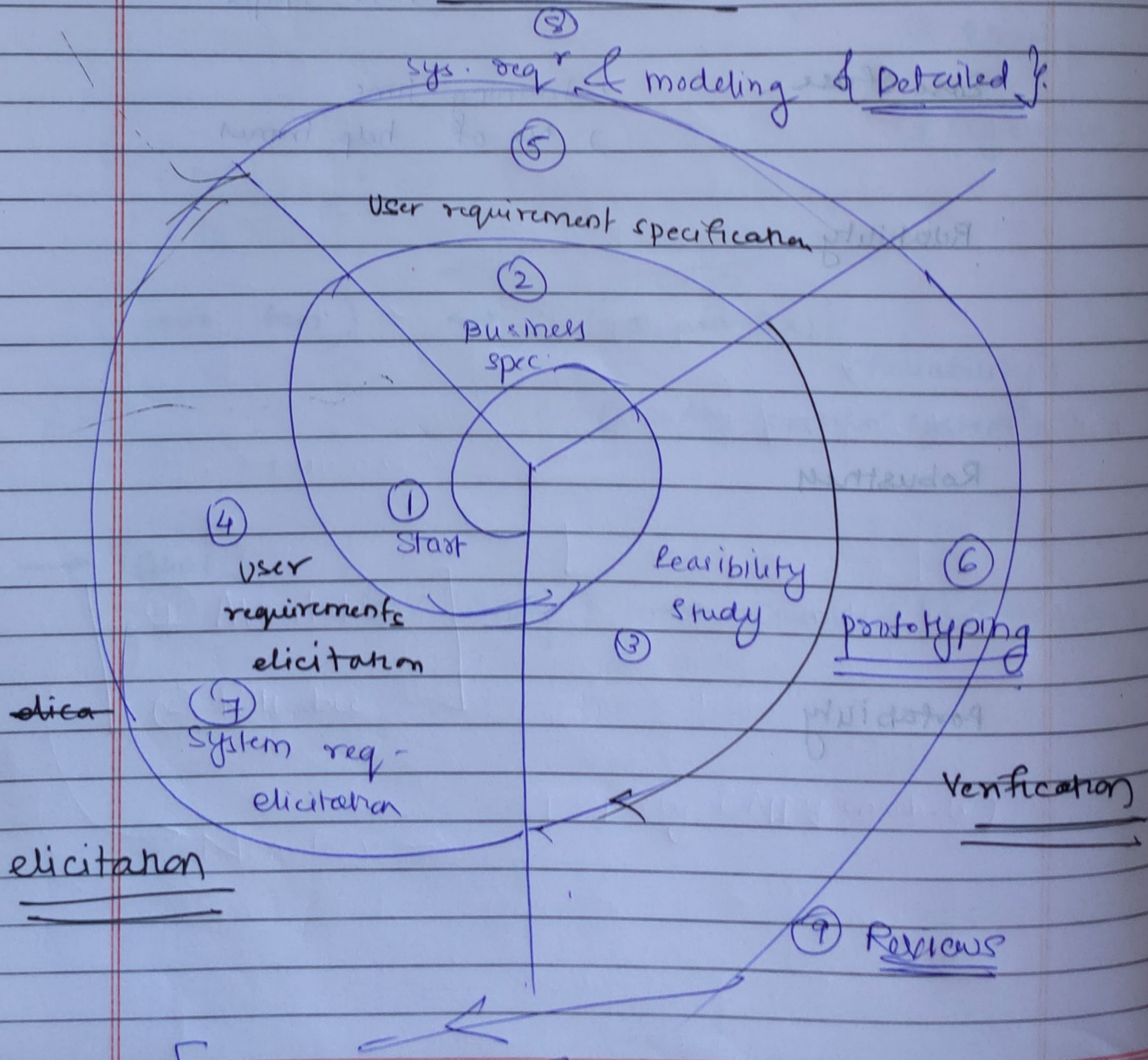
Robustness

Portability

LIBSYS

- Natural language spec's
- semi-formal spec's
- formal spec's

Specification



✓ Natural Language

- Paragraph
-

(Structure NL)

[UML Diagrams]

Graphical } Semi-formal

✓ Mathematical Specification

really difference { requirements & Design }

(A) Natural Language specification.

(fcx) + Nonfcx)

① Lack of clarity
② requirement confusion
③ Requirement amalgation
(Authentication)
↳ Non-fcx)
(security)

(C) Case study ① Mentcare System

② Insulin Pump control system

H/w → insulin pump (S/w)
↓
[compute dose of insulin]

① conditions.

[Low blood ~~sy~~ sugar]
- Brain malfunction
- coma
- death

[① Available
② Reliable]

→ Non-fcx) ⇒ property (Apply on whole)
✓

Card Based spec's

Insulin Pump / Control Software / 3.3.2

Function

Description

Inputs — current sugar reading (r2) — prev readings r0, r1

Source — current sugar reading from system

Output — CompDose

Destination — (Main control loop)

Action → tabular

Condition	Action

Requirements

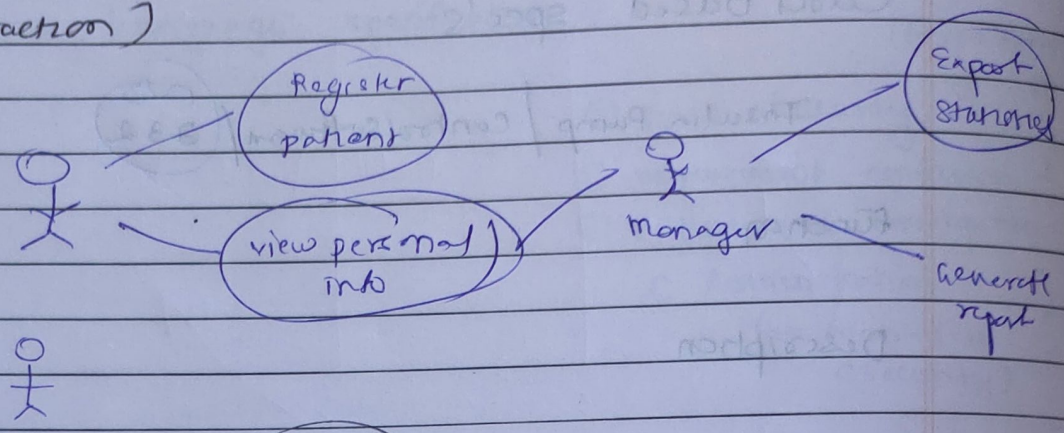
Pre-conditions

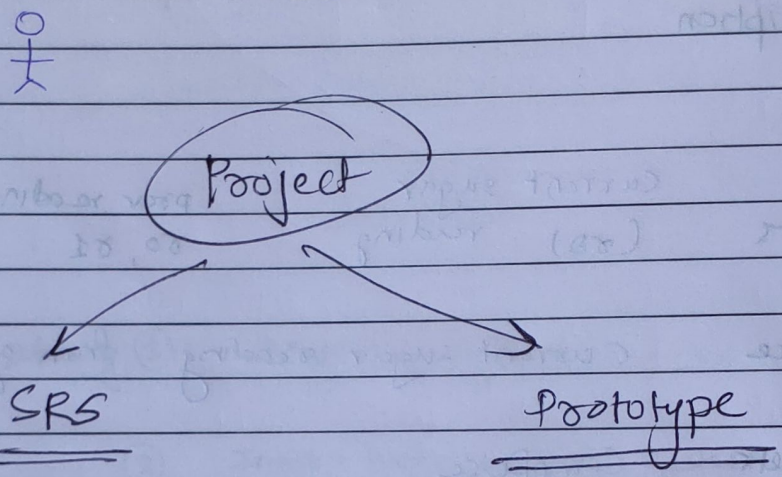
Post-conditions — (r0 ← r1 r1 ← r2)

Side-Effects

UML Diagrams

(Interaction)





✓ [IEE Standard]

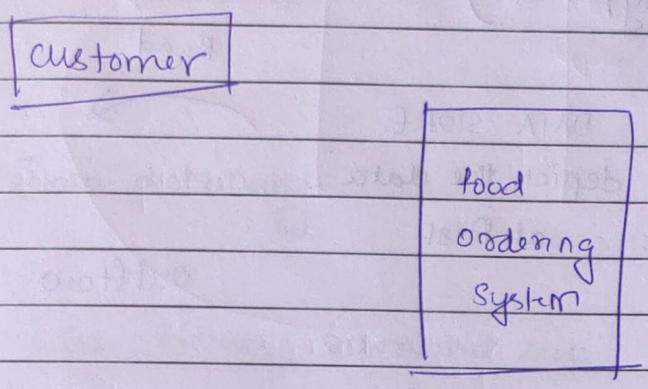
1. Preface
2. Introduction
3. Glossary
4. User req
5. Sys Architecture

Zero

(Level 0)

(*) Context Diagram for fast food IS

decomposition ✓
integration X



(Decomposition)

(1) no. of inputs & outputs must be same

✓ Black Hole

✓ Mirade

✓ Gray Hole

DFD's

Date
Page

(1)

(Create/modify/delete)

write → DATA STORE → Read

depict the data at Rest

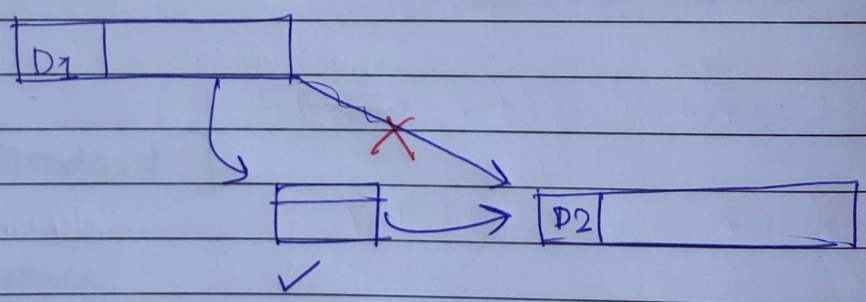
inflow

outflow

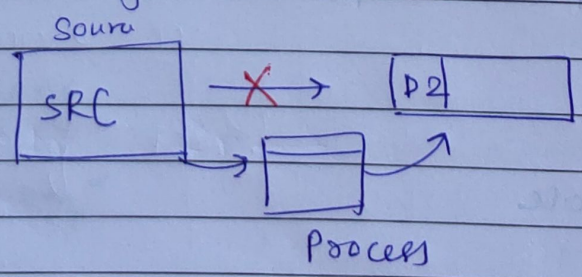
(2) Rule for Data stores

DS = Datastore

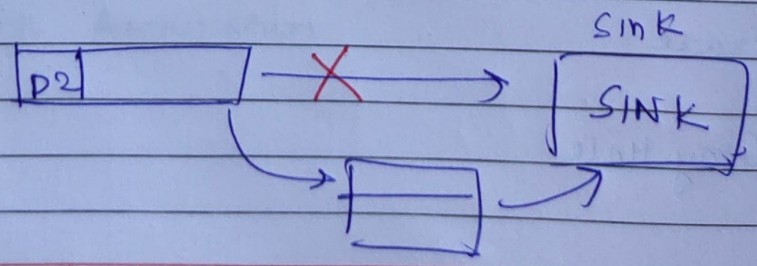
(1) data cannot move (1DS) — (Another DS)



(2) External agent should not directly interact with DS



(3)



✓ Naming Rules

✓ Balancing Principle = no. of input before decomposi

= no. of input after decomposition

③ Data dictionary

(description of data).

Eg. Railway reservation sys

Passenger → name { first, middle, last

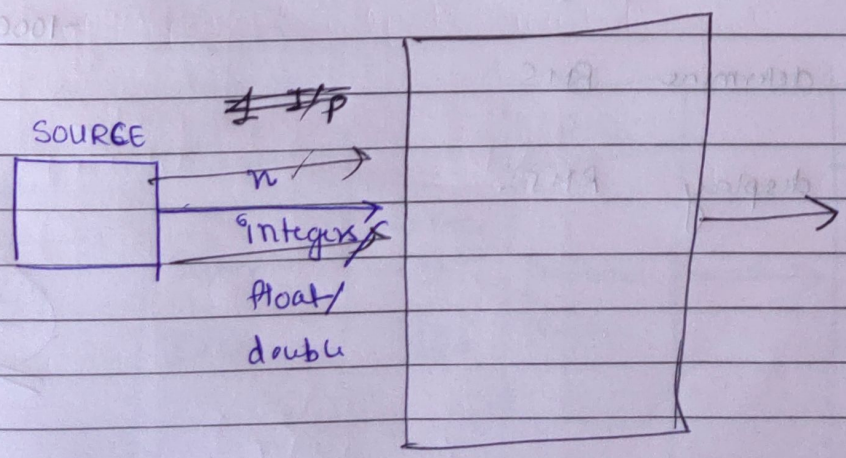
→ address { local → House no, street, city, state

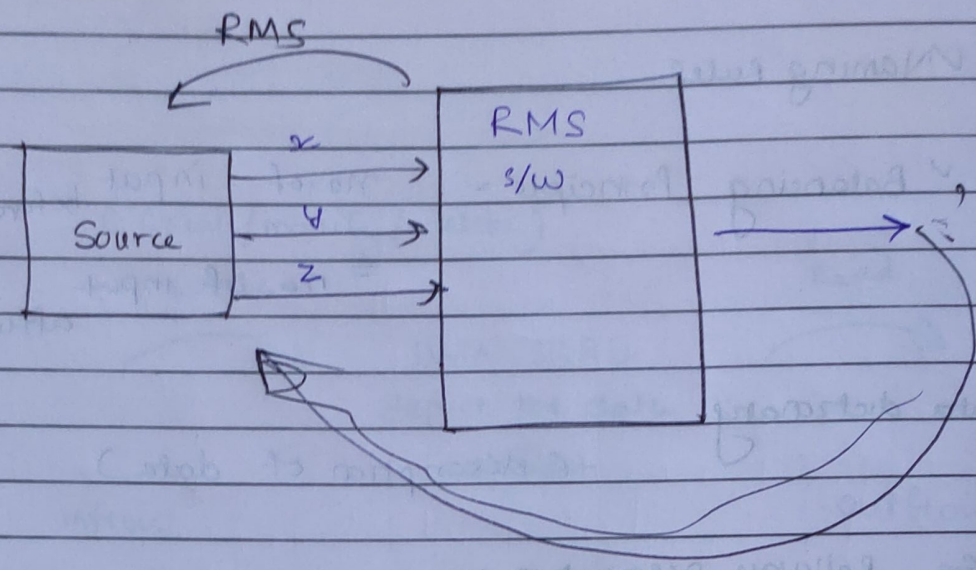
work / community addr

zipcode

(Q) (RMS software)

root mean square





① read 3 int's x, y, z $-1000 \le x \le 1000$

~~$-10 \le y$~~

① read 3 int's

② validate within range

requirements

① Read three integers from user in the range of -1000 to 1000

② determine RMS

③ display RMS.

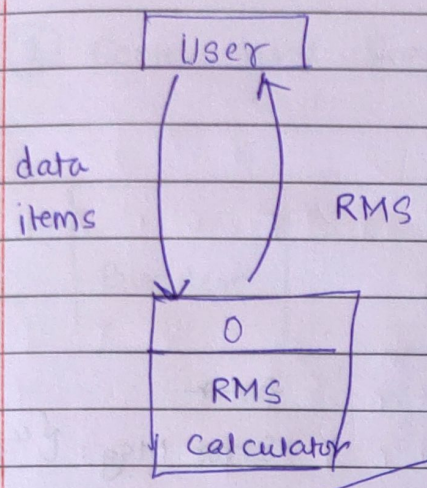
$\frac{10 \times 3}{3}$

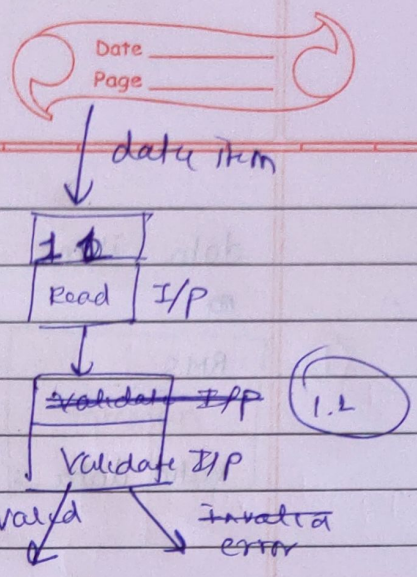
1.0 2.0

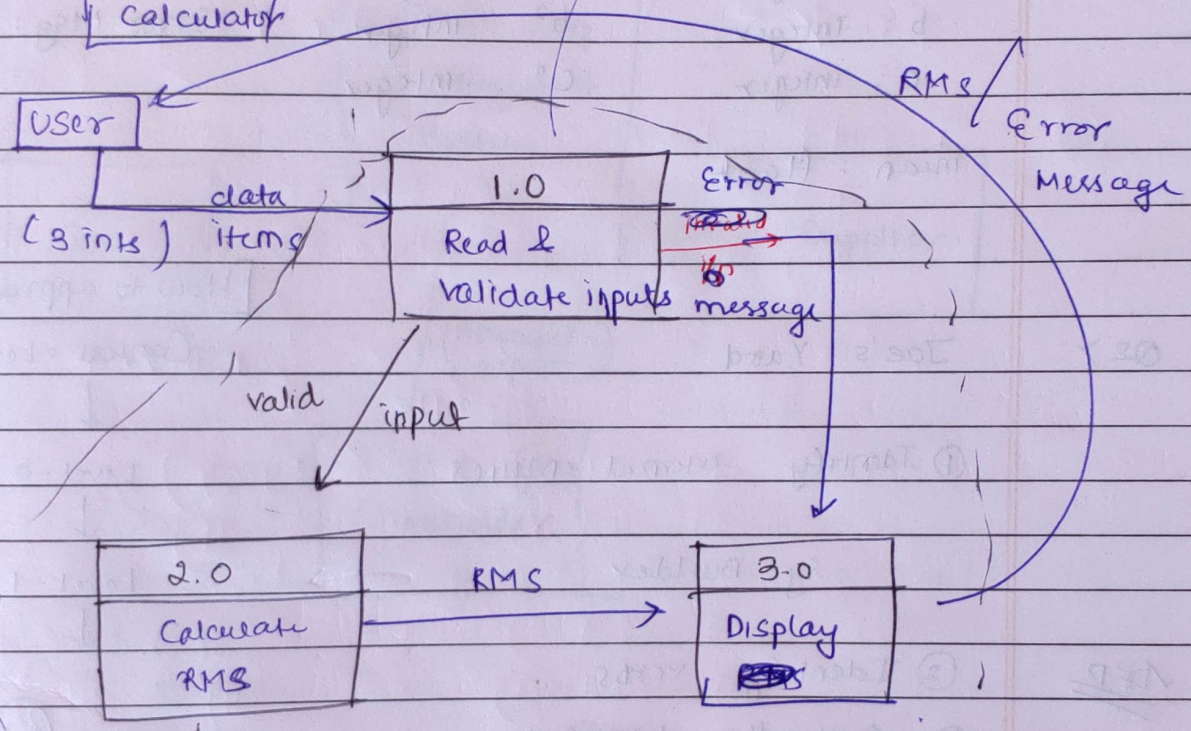
1.1 2.1 Sqr

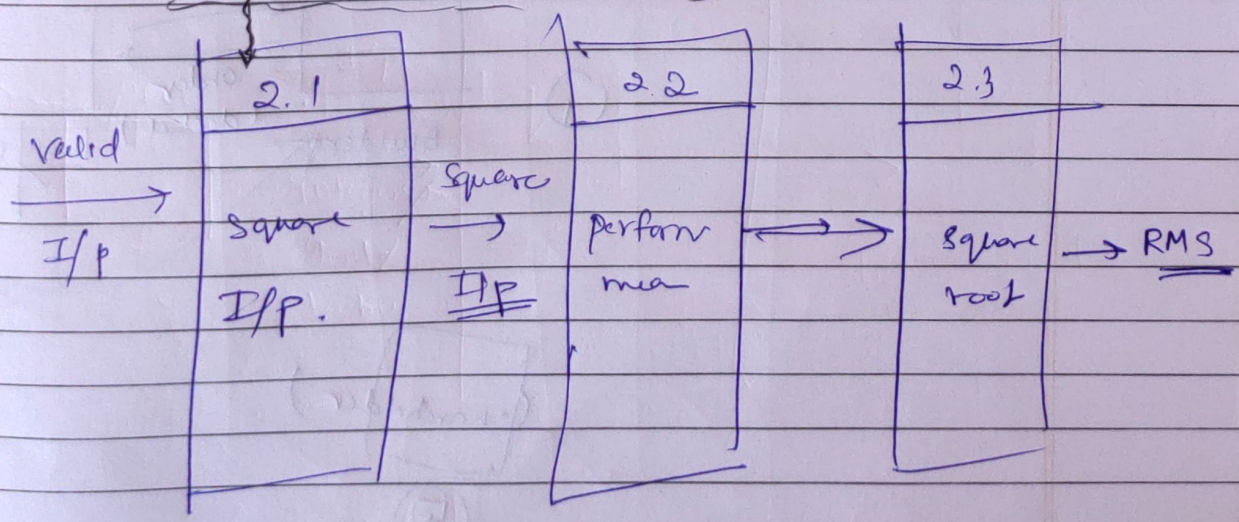
1.2 2.2 mean

2.3 roots









data items : {integer} 3

RMS : float

valid data : data items

a : integer	a^2 : integer	~~Enter~~
b : integer	b^2 : integer	Error Msg : {"invalid Inputs please try again"}
c : integer	c^2 : integer	

mean : float

[How to approach?]

Context - Level

↓

Level - 0

↓

Level - 1

↓

Q2 > Joe's Yard

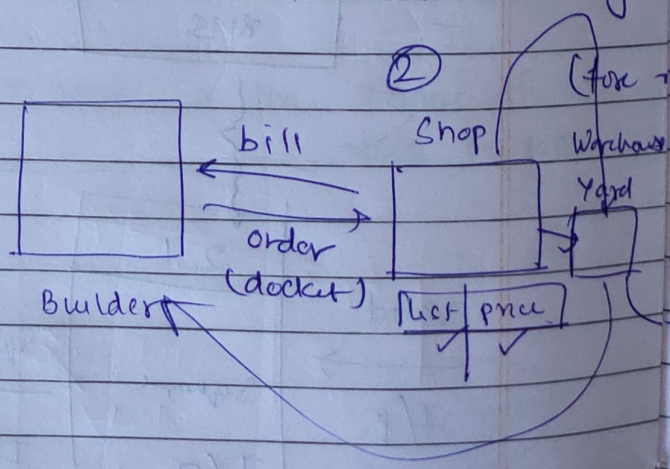
① Identify external entities

Eg: Builder

DFD

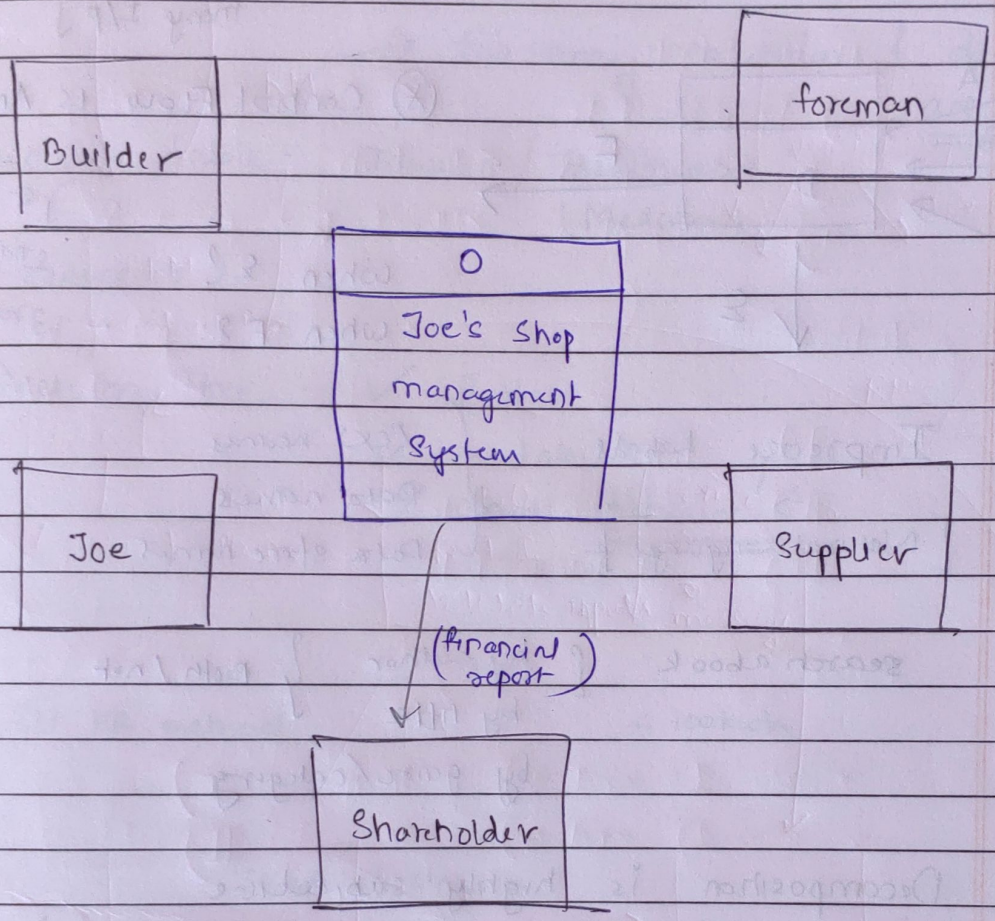
② Identify verbs

③ Group the processes

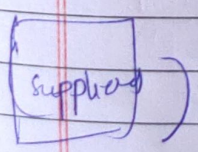


(shareholder)

① Context-level Diagram



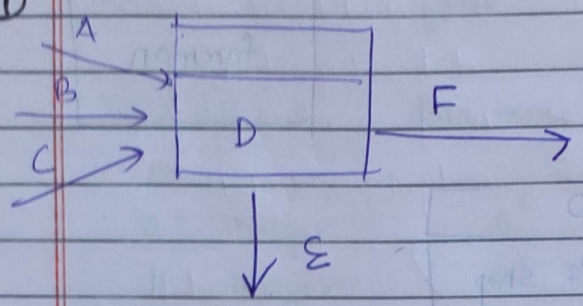
② [Process Decomposition]



④

* Shortcoming of DFD Model

How many I/P } → { How many O/P

① 

(*) Control Flow is Ambiguous

when E? — 1st op, 2nd op

when F? — 3rd op

② Imprecise Labels [Natural language]
- Fcx) names
- Data names
- Data store names

search a book { by author, by title, by genre/category } Both / not

③ Decomposition is highly subjective

(Mathematical) (Eqn) = Formal

DFD = Semi-Formal

= In-formal

structural tree

&

Logic Modelling

Date
Page

① DFD
②

too many conditions & decision actions

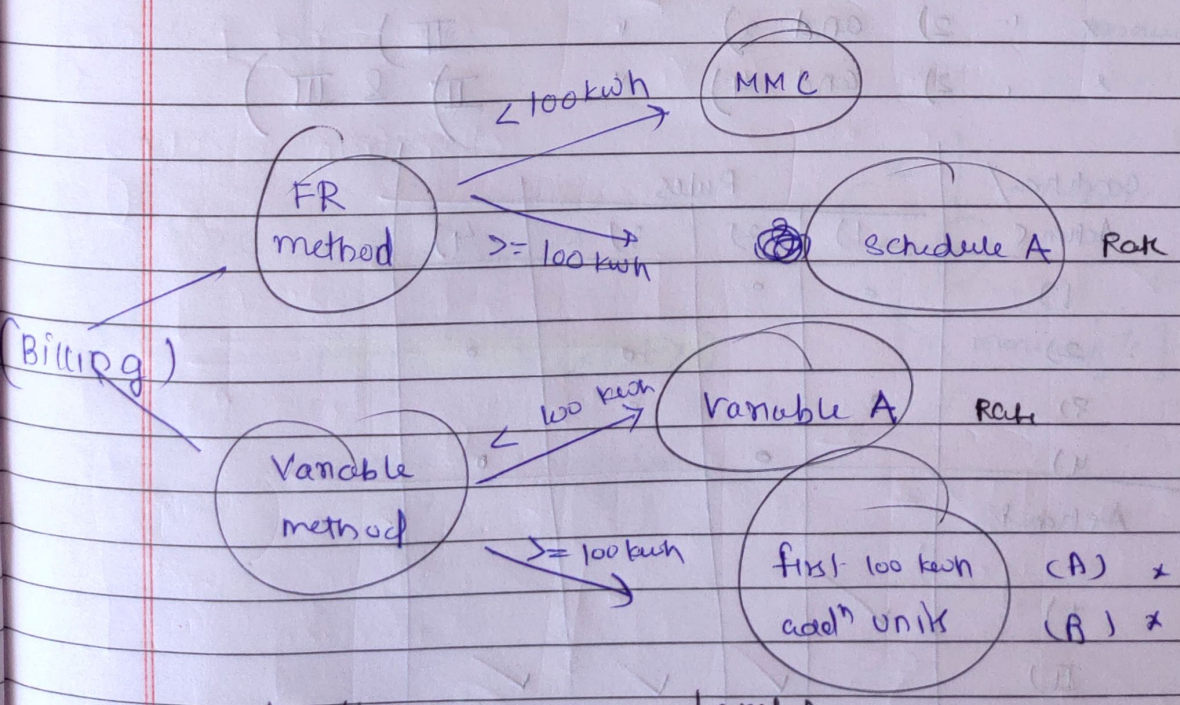
(I) Decision table - Insulin / Healthcare (Medcare)
(reduce it)

(II) Decision Tree
✓ Better
✓ readability
✓ small decision's

(A) ① FR method

MMC
< 100 kwh

Schedule A structure



Conditions

1) Fixed Rate
2) Variable Rate
3) consumption < 100 kwh
4) consumption ≥ 100 kwh

Actions

I) MMC
II) Schedule A Rate
III) Schedule B Rate

Rules

* If 1) and 3) then I)
* " 1) and 4) " II)

* " 2) and 3) " II)
* " 2) and 4) " II) & III)

Condition/ Actions	Rules			
	1)	2)	3)	4)
1)	o	o		
2)			o	o
3)	o		o	
4)		o		o
Actions				
I)	✓			
II)		✓	✓	✓
III)				✓

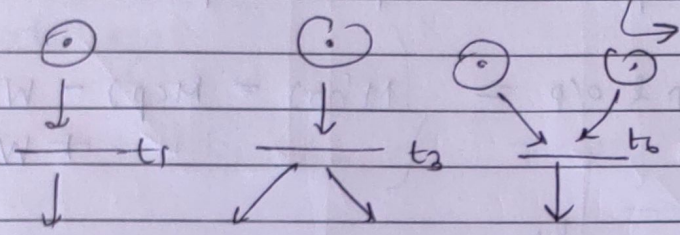
→ DFD ✓
→ Decision Tree ✓
Petri Net

Date 30 Aug
Page

Modelling

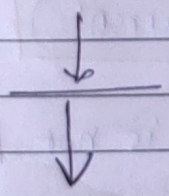
(A) Petri Net → specify concurrent situation (Starvation / Deadlock).

✓ Process ○
✓
✓ Transition
✓ Marking (availability of resource)



SRS
✓ (Pre-condition)
{ Event }
(Post-condition)

* [no. of marking ≥ weight of transition]
✓ { for all input places }

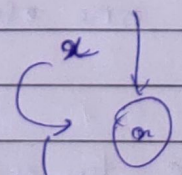


{ ✗ Not consumer & producer }

∀p ∈ t's input places,

$M(p) \geq W(\langle p, t \rangle)$

marking weight



['x' marking 's]

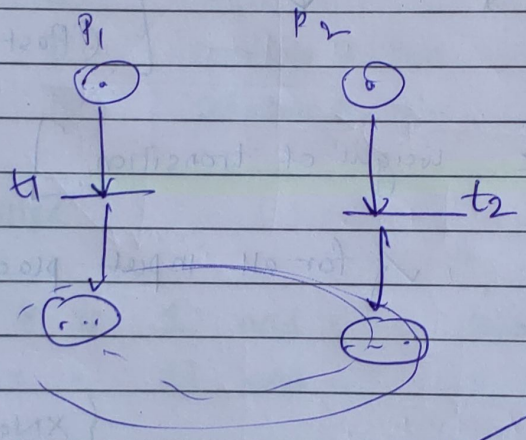
t fires

p is I/p place = $M'(p) = M(p) - W(\langle p, t\rangle)$

O/I place = $M'(p) = M(p) + W(\langle t, p\rangle)$

both I/p & o/p = $\left\{ M'(p) = M(p) - W(\langle p, t\rangle) + W(\langle t, p\rangle) \right\}$

Eg:



t_1 ✓
t_2 ✓
t_1 & t_2 ✓

(t_1 & t_2)

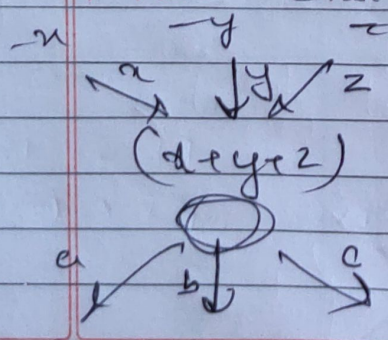
(Concurrent) independent

can fire simultaneously/ one after another

one is not preventing other from executing

✓ * Helps to check for [starvation & deadlock]

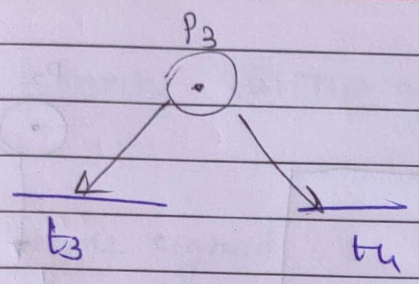
* Note: Conservation of Marking is NOT followed



* [Ambiguity in removal of marking]

$a+b+c \neq x+y+z$

Date 21 sept
Page



Shared resource
→ execution of one prevents other to happen
→ (mutually exclusive)

(D) Modelling

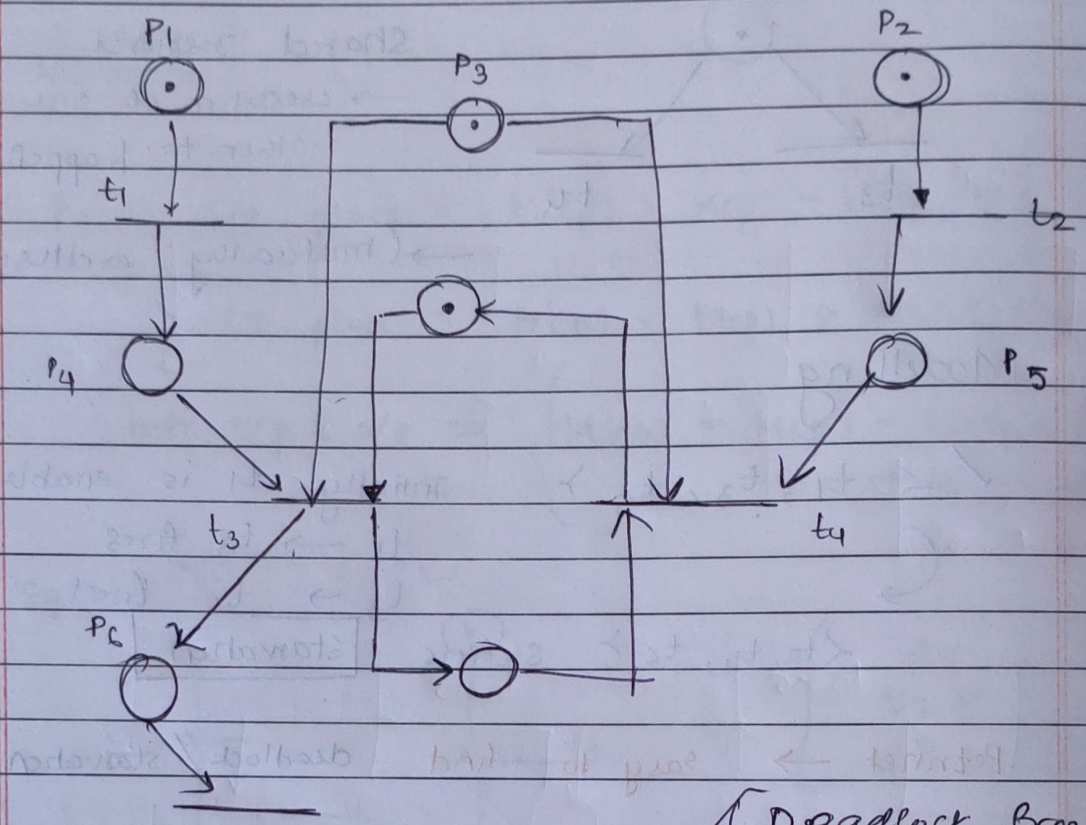
✓ $\langle t_1, t_3, t_5 \rangle$ Initially t_1 is enabled
$t_1 \rightarrow t_3$ fires
$t_3 \rightarrow t_5$ fires

$\langle t_2, t_4, t_6 \rangle$ suffers $\boxed{\text{Starvation}}$

Petrinet → easy to find deadlock / starvation

Common Cases

- concurrent: t_1 & t_2
- conflict: t_3 t_4
- firing sequence: $\langle t_1\ t_3\ t_5 \rangle$



✓ (Deadlock free)
= Live

✓ Starvation

t_1, t_3, t_5, t_2, t_4

alteration in process execution

$t_3 \longrightarrow t_4 \longrightarrow t_3$

Deadlock

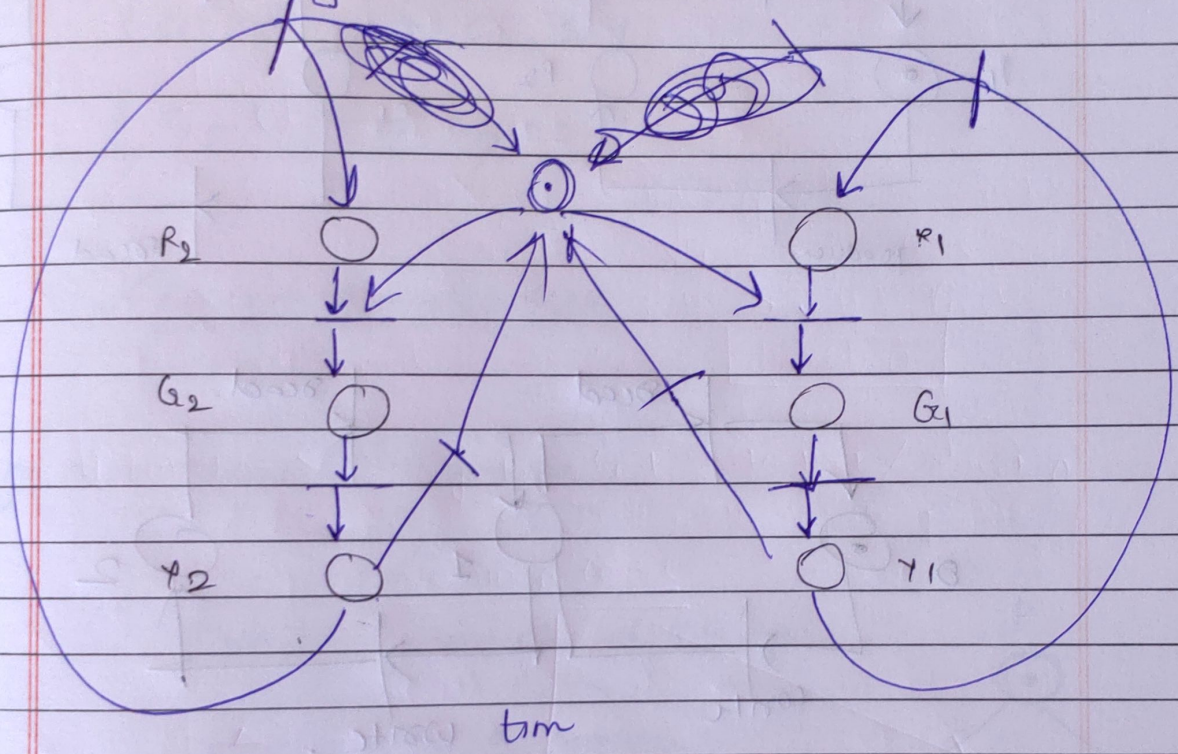
$\langle t_1, t_2, t_3', t_4' \rangle$

$\{t_3', t_3''\}$ (critical section)

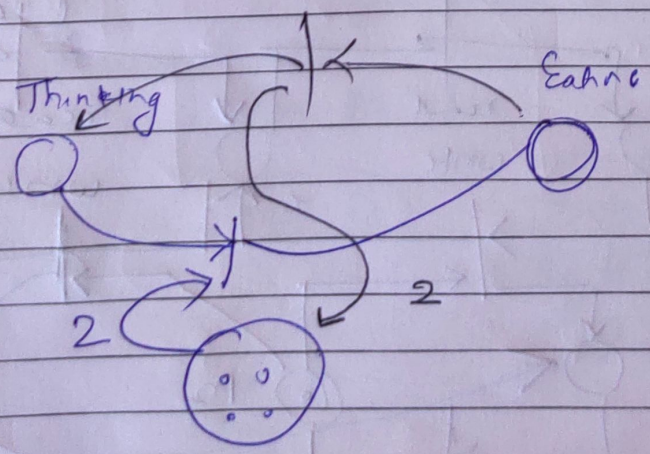
[① Priority ② Timing]

How to judge petrinet is efficient?

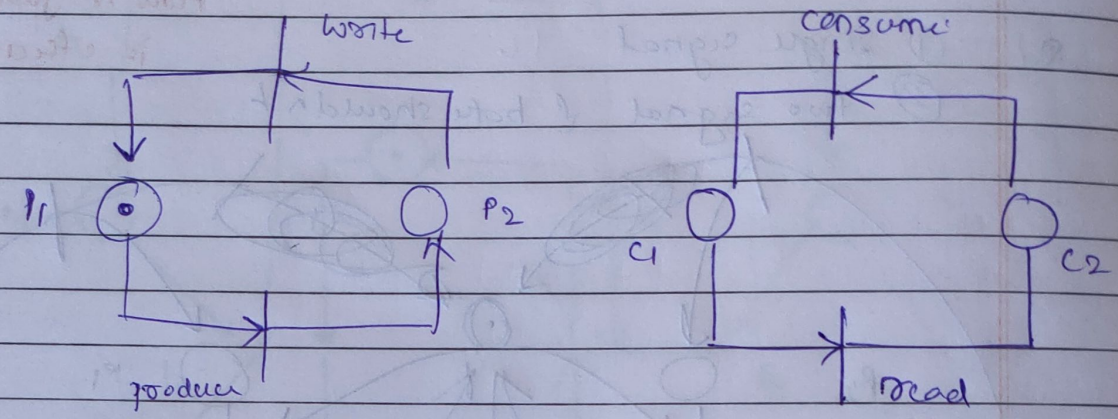
Q) ① Single signal
② two signal & both shouldn't

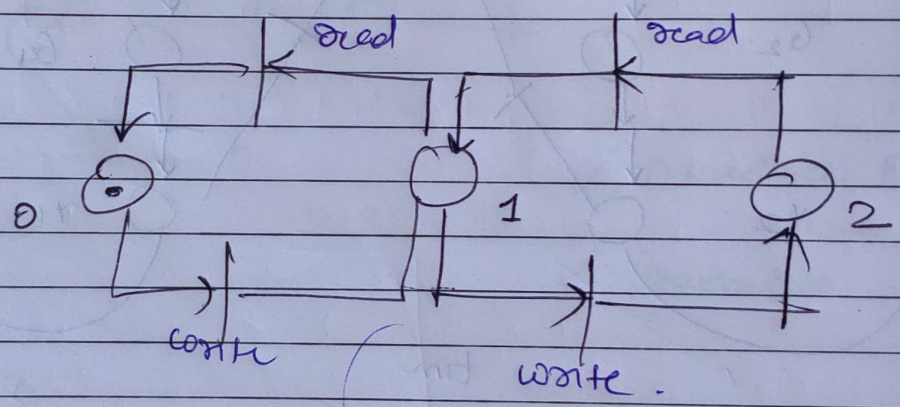


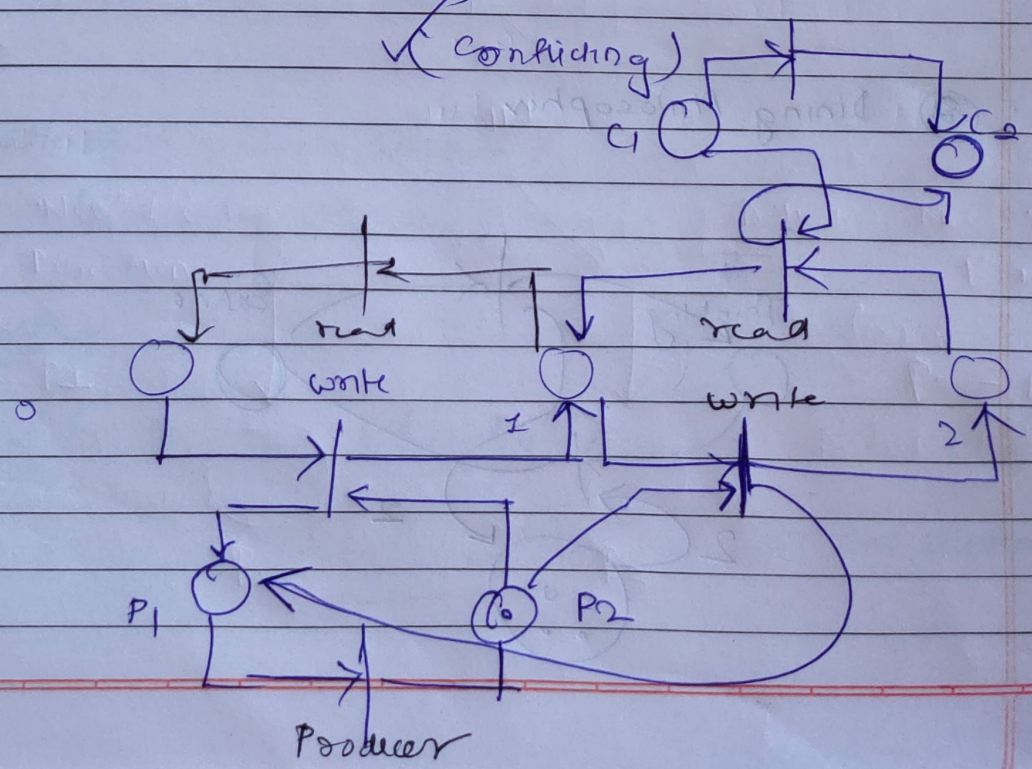
③ Dining Philosophers



(c) Producer & consumer problem







Date
Page

Rules

$\{P_2, 1\} \rightarrow P_1, 2\}$

buffer

$\{P_2, 0 \rightarrow P_1, 1\}$ write ✓

producer

$\{c_1, 2\} \rightarrow c_2, 1\}$

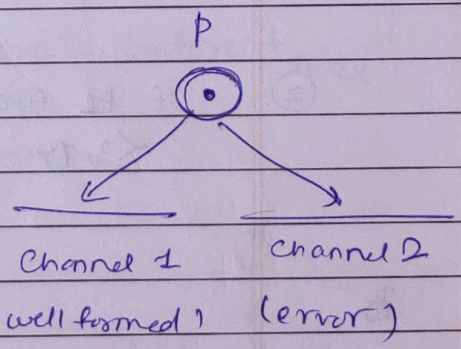
$\{c_1, 1 \rightarrow c_2, 0\}$

✓ Concurrent activities

✓ {limitations & extensions}

① Check the content of message

• → existence of resource



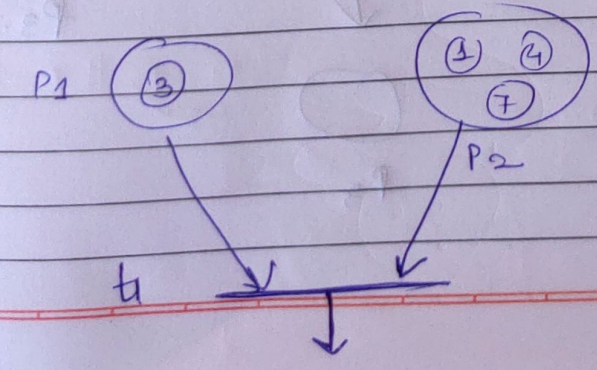
Extension 1

Sol^n ⇒ Predication
(eq^n that must be satisfied to enable it)

function

$P4 := P1 + P2$

as with t1



P_1	P_2	enabled
3,	4	✓
3,	7	✓

→ ready tuple

✓ {atleast 1 ready}

	t_1 (enabled)	t_2 (enabled)
Ready tuple	<3,4> <3,7>	<4,4>

(t_1 fires) { any one of ready tuple }

↳ <3,4> ⟹

$P4 := P1 + P2$

7 ✓

& Don't Forget to remove

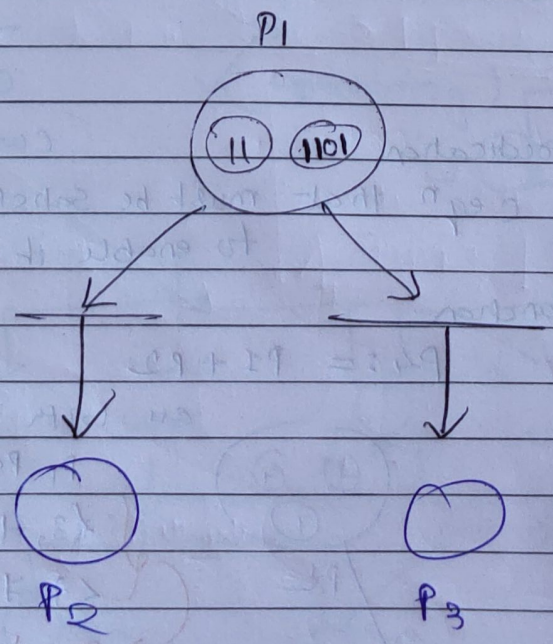
< 3 , 4 >

P_1 P_2

① if t_1 fires → t_2 can't fire
<3,4> { t_2 not enable }

② If t_1 fires → t_2 can't fires
<3,7>

To



Predicate: No of 1's in P are even

fcx): $P_2 := P_1$

Predicate: No of 1's in P is odd

fcx): $P_3 := P_1$

Extension 2: specify priorities of transition

★ transition having ↑ higher priority will fire first

Extension 3: Timed Petri Nets

$\langle t_{min}, t_{max} \rangle$

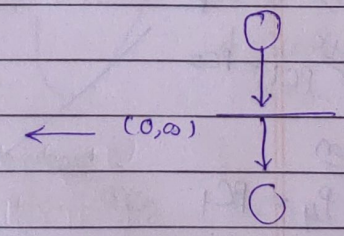
↑ $\langle 2, 10 \rangle$

[enabled + only fire after t_{min} sec & before t_{max}]

after it is enabled wait for t_{min} sec to fired

fire before 10 sec (t_{max})

(a) t_1 & t_2 enabled



event can be executed any time

[$\langle 0,0 \rangle$ → as soon as it satisfied, it should fire]

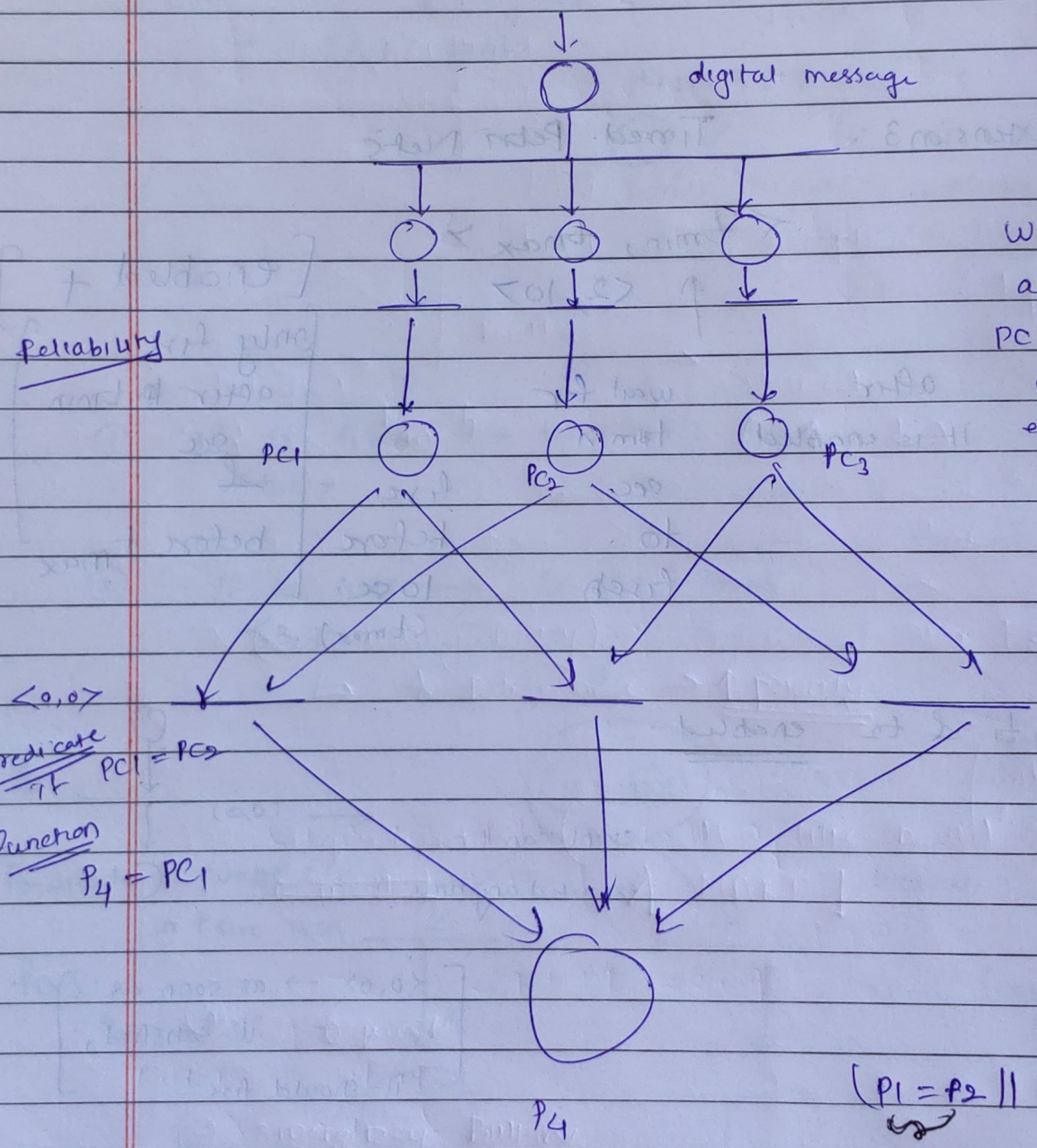
Date 6 Sept
Page

Case 1: ① Message must be triplicated

② 3 copies → 3 diff channel

(AMBIGUOS)

③ receiver accepts based on [2 out-of-3 voting policy]



What if all 3 PC1, PC2, PC3 are enabled?

(P1 = P2 || P2 = P3 || P1 = P3)

✓ (comparisions more)

Predicate $PC_1 = PC_2$ or $PC_2 = PC_3$ or $PC_1 = PC_3$

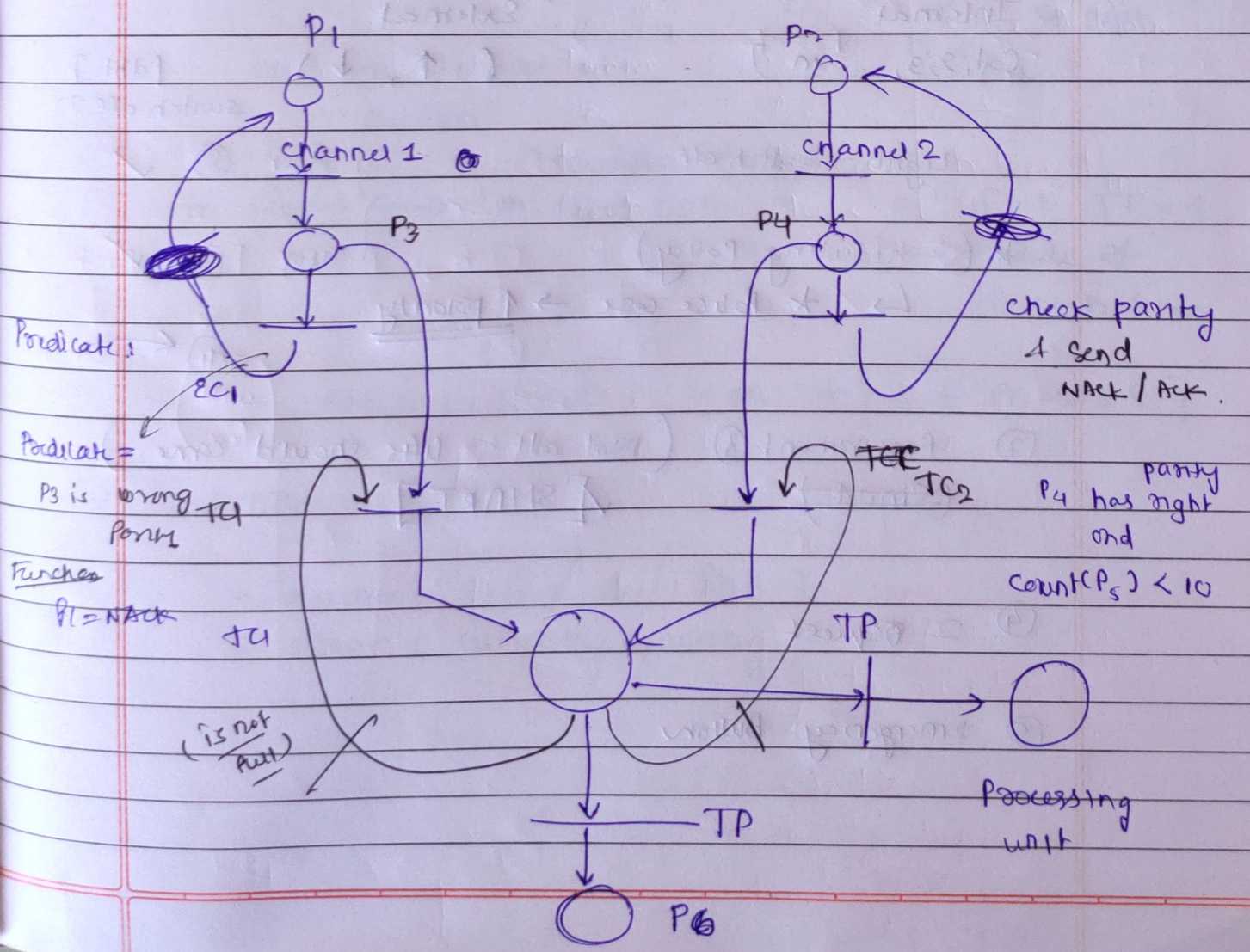
Function if $(PC_1 = PC_2)\ ?\ PC_1 : (PC_1 = PC_3)\ ?\ PC_1 : PC_2$

(Q) PN extension

① 2 diff channels

② checks parity of each message

③ if parity (wrong) — NACK via reply channel
(right) — received message in Buffer



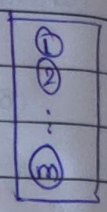
Last topic of Petri Net

Case Study : Elevator system

(A) Requirements

① n- elevator system

② m- floored apartment → M floor



① Green light switch off

② floor - [↑] [↓] } exception { ground } [↑] { top most } [↓]

Button

Internal (1, 2, 3, .. m)

External (↑, ↓) [dbt] switch off ??

Light - switch off

floor ⑧ ✓

floor ⑤ ✓

★ (Scheduling Policy)

↳ ★ Police case → ↑ priority

④ ←

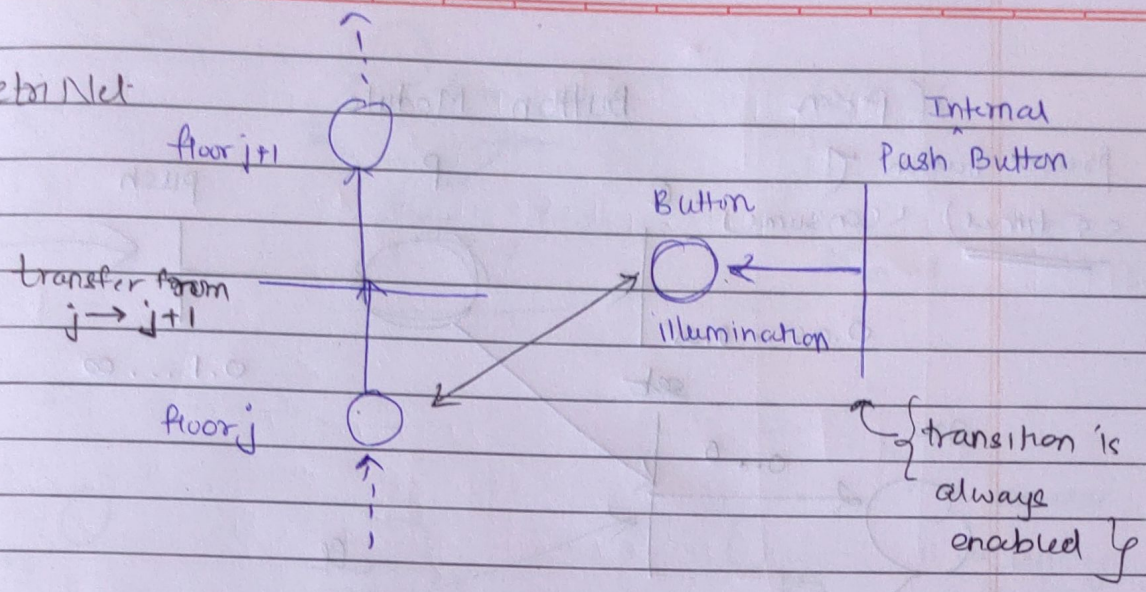
③ Requirement ③ (Not all lifts should come)

~~(smart)~~ ✓[SMART]

④ 2 request

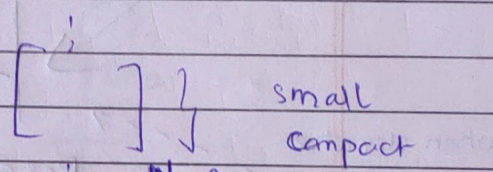
⑤ emergency Button

Petri Net



Modularized Approach

① * ELEVATOR_BUTTON



Petrinet model of
Button

m floors → 'm' buttons

2 modules of type

GF = 1 TF = 1
2 module of
on each floor

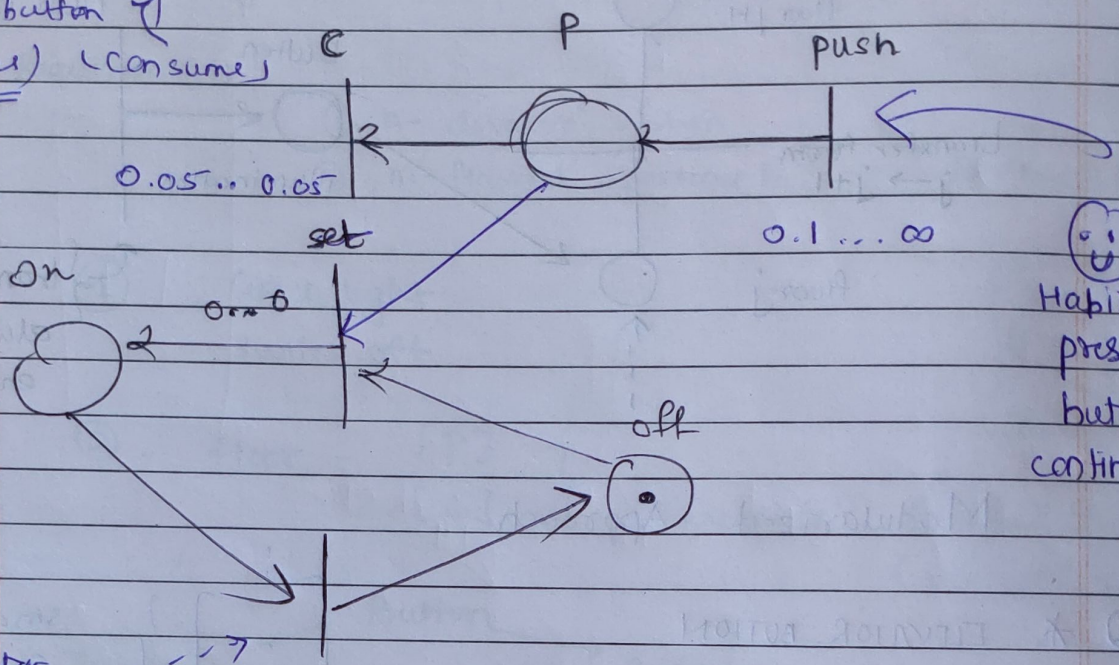
total buttons $= m + [2 + (n-2) * 2]$

② ELEVATOR_POSITION

- direction (↑ / ↓ / still)
- floor (currently passing)

if person presses button (5 times) (consume)

Button Module



Habit of pressing button continously!

(when elevator reaches floor)

(*) **Question** What is meaning of transition sequence ??

off, on, same floor, off

< Push, set, Push, Push, Reset, Push, Push, ~~set~~ >

✓ ✓ (e) (e) consume by

set will fire

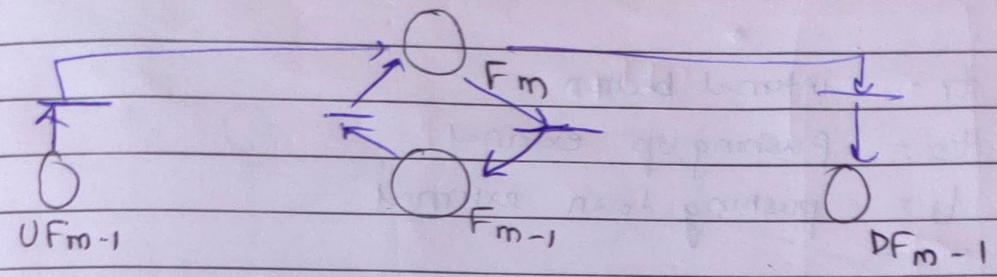
(Cannot fire)

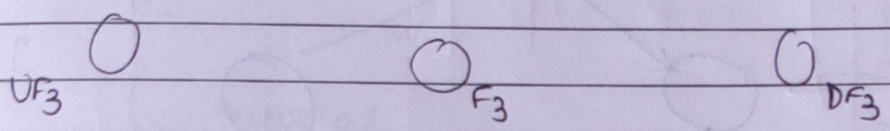
(consume)

Date 13 Sept

Page

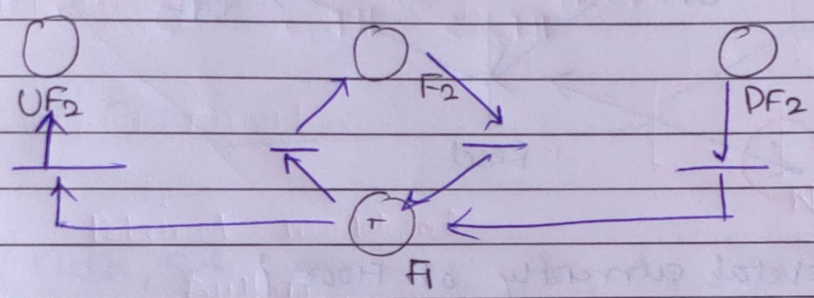
Elevator Position





∘ → elevator is currently at floor

$(>2 <= m)$

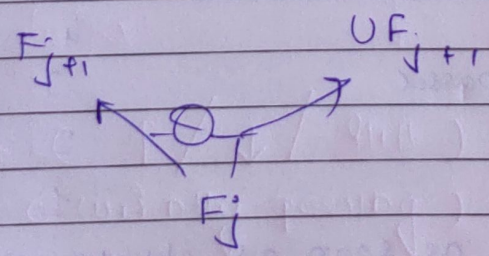


UF_2 = upward direction, passing through 2.

temporary transition →

Imp

① time delay for F_j to $F_{(j+1)}$ / $UF_{(j+1)}$



✓ 10

② if request for UF_{j+1}, but someone presses button for

✓ ('6')

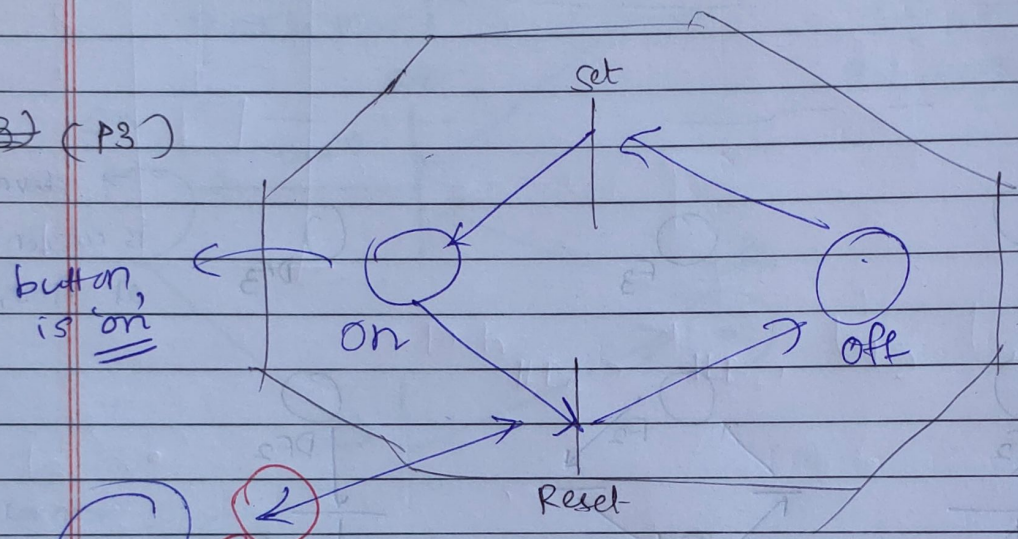
✓0 ← 6

5

t_1 : internal button
t_2 : pushing up external
t_3 : pushing down external

(15-Sept-22)

(~~Q3~~) (P3)



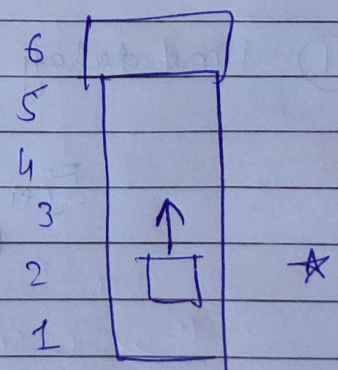
E_j

Elevator currently at Floor j

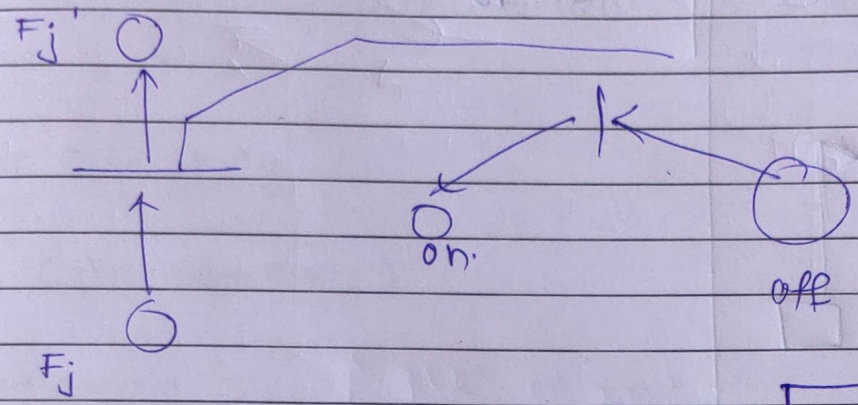
why!? Elevator is still a floor 'j'
↳ token should not be removed since

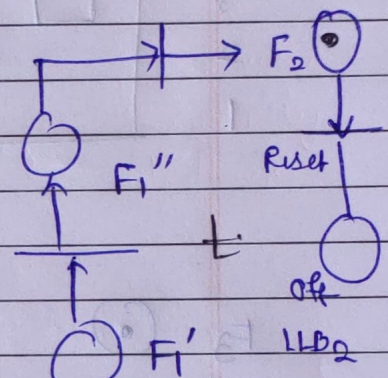
(Q) (UP_5)

After it passes



off → as soon as elevator reaches floor 'j'
→ floor 'j' & desired direction
↳ after 'x' seconds

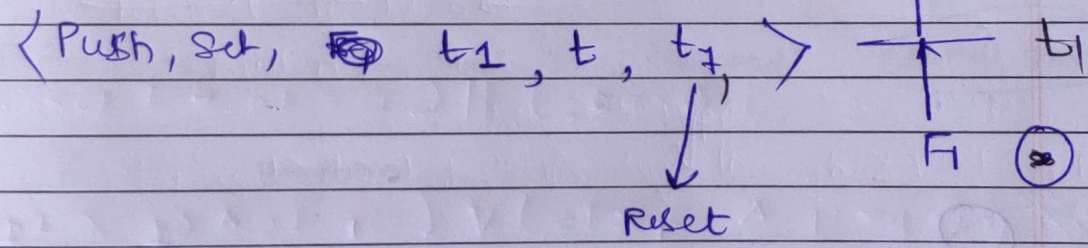


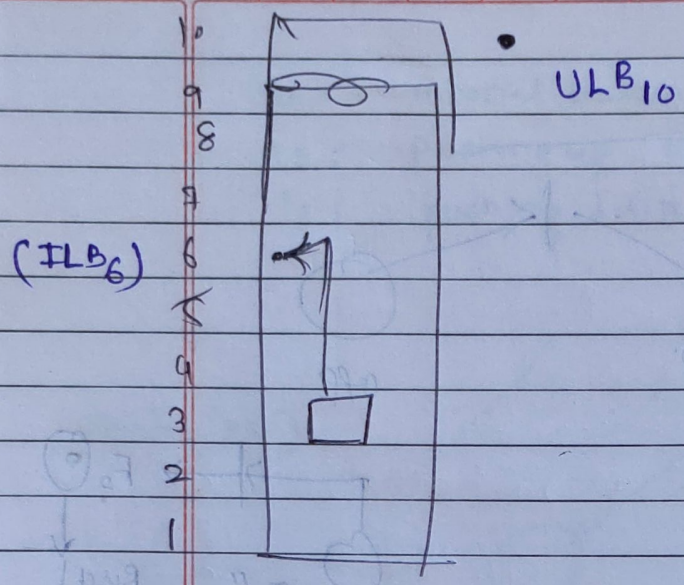


tutorial

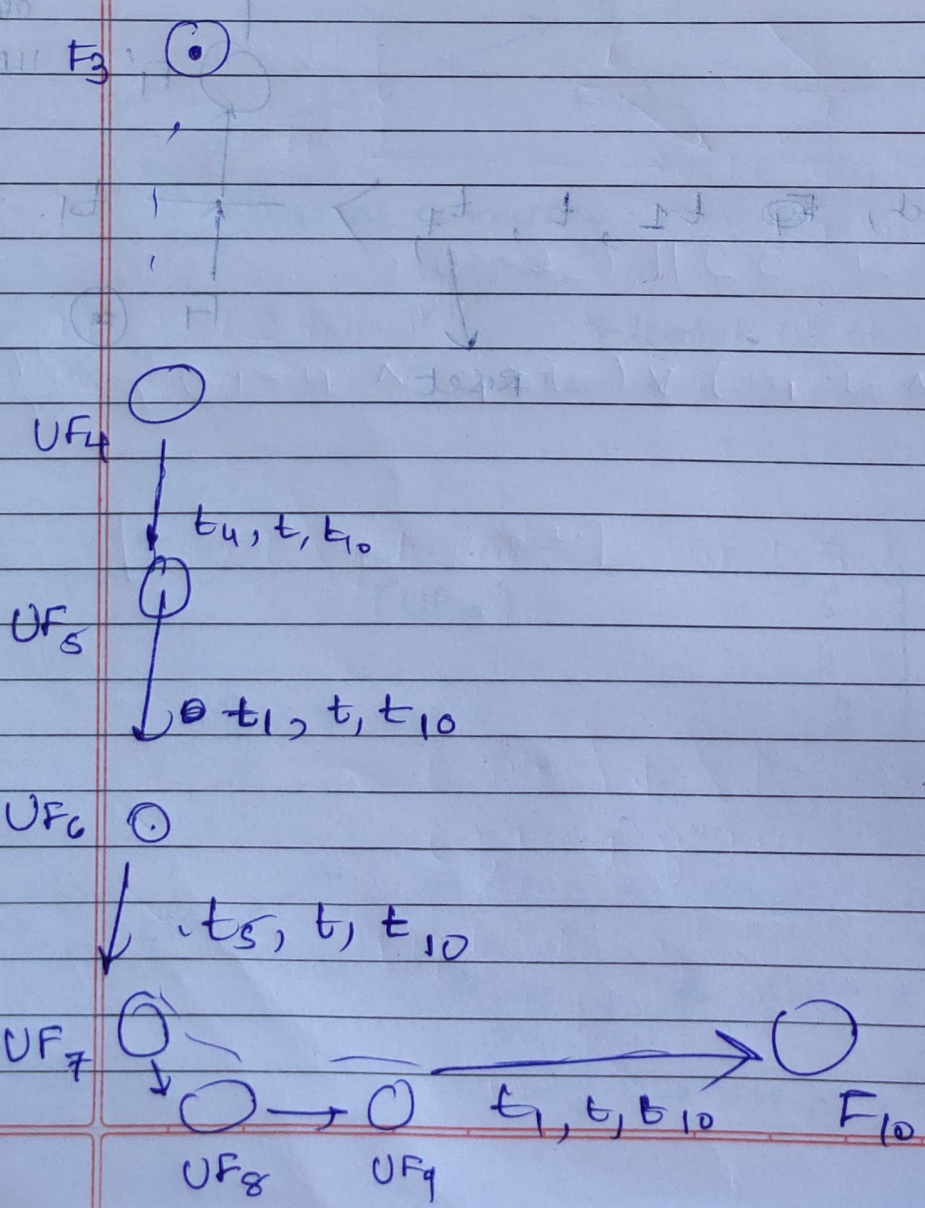
(Q)

ILB_2





$\langle Push_{ILB_6}, Set_{ILB_6}, t_4, t, t_{10}$



Date 20/9
Page

Logic Specification ← includes (descriptive specification)

① Elevator Case Study

① FOT (First order Theory)

Address saving

name != " " ∧
state in statelist ∧
zip >= 10000 && zip <= 99999 ∧
street != " "

{ Pre (i_1, i_2, ---- i_n) } Pre-cond^n → Initial state

P ← program

{ Post (o_1, o_2, ... o_m, i_1, i_2, ... i_n) } Final state

[Input Output Assertion]

Eg: Requirement for Division

{ exist z (i_1 = z * i_2) }

P

{ o_1 = i_1 / i_2 }

output inputs

Better ✓

$\{ i_1 > i_2 \}$

P

$\{ i_1 = i_2 * o_1 + o_2$ and $o_2 >= 0$ and $o_2 < i_2 \}$

→ less restrictions on I/P ↑

→ ~~more~~ more restrictions on O/P ↑ [more precise]

✓ $\{true\}$ // No restrictions on User

P

$\{(o = i_1$ and $o >= i_2) \| (o = i_2$ and $o >= i_1)\}$

Right ✓ $\{(o_1 = i_1 \wedge o_1 \geq i_2) \vee (o_1 = i_2 \wedge o_1 \geq i_1)\}$

[Sum of I/P sequence]

$\{n > 0\}$

P

$$\{ o = \sum_{k=1}^{n} i_k \}$$

Date
Page

(50)

(Q) GCD of two numbers i_1 i_2

7 13

gcd(x, y)

~~y x.y~~

$\{ i_1 >= 0 \wedge i_2 >= 0 \}$

P

exist z

$\{ O =$

$\{ i_1 > 0 \wedge i_2 > 0 \}$

P

exist z i_1/z i_2/z

Not exist y i_1/y i_2/y $y > z$

$O = z \wedge$

$\{$ exist z, k_1, k_2 ($i_1 = z * k_1 \wedge i_2 = z * k_2$)

$\wedge$

$\boxed{\text{not exist } y}$, ($i_1 = y * k_3 \wedge i_2 = y * k_4 \wedge (y > z)$)

, (exist k_3, k_4)

(*) WAP to reverse i/p sequence

$\{ I_1, I_2, \ldots I_n \}$ $\{ n > 0 \}$

n P

for all i ($1 \le i \le n$) $\{ O_i = I_{n-i+1} \}$

$O_i = I_{n-i+1}$ $1 \le i \le n$