Principles of Programming Languages

Program:-

→ Eiffel

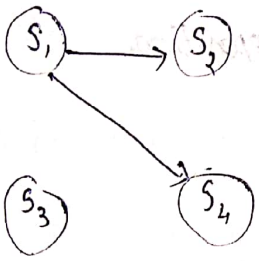
→ 700 Programming Languages

Randomness

Then Probability distribution should be discussed

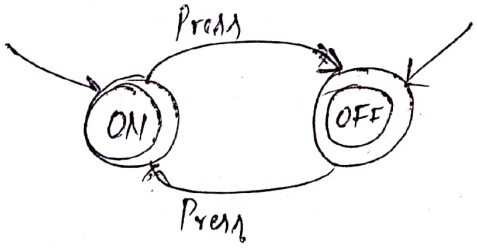
Non-Deterministic

No need of discussion of Probability distribution

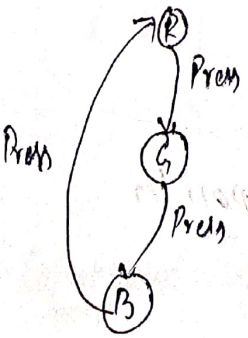


State Transition.

For a switch,

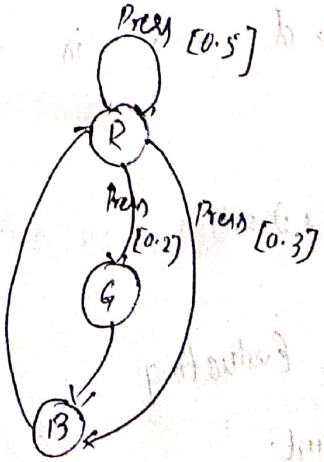


Special switch R, G, B,



In case of Non-deterministic

Non-Deterministic



Preposition :-

No necessitity to be true.

No need to prove.

For a Preposition, subjectivity should not be involved.

Eg :- Food in BBQ is good.

→ Universal Truth (or) false

a[100]

BFS is better in DFS

Atomic Proposition

Preposition that cannot be broken into other proposition.

a ≡ It is raining today } Preposition.
b ≡ Roads are wet today }

a → b ≡ Proposition

↳ Truth value = 1

c ≡ 2 + 2 = 5
d ≡ Buffaloes can fly. } c → d is a preposition

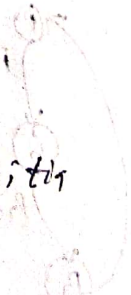
c → d → is true.

d' ≡ 2 + 2 = 4 } c → d is a preposition

↳ is TRUE

Context of Evaluating is important.

d → c cannot be evaluated



If p,q are atomic propositions,

$p \vee q, p \wedge q, \neg p$ → are prepositions.

Predicate

Def:- Mathematical function with co-domian True / False

→ Are all prepositions, predicate?

Yes. As it's co-domain is True/False.

Is converse true?

No. Subjectivity can be involved. (Free variable → for prepositions.)

Eg:- $x > 2$ $\quad x \in Z$

It is a predicate.

It is not a preposition.

$f: Z \to \{True, False\}$

$x \in Z \quad x^2$

$f(x) = x^2 \quad f: Z \to Z$

Free Variables:-

$x + y = 2$ → It is a predicate

x, y are free variables.

Constraint variables:-

operator

$\exists x.$ logic in preposition but not predicate.

$\exists x . x = 2$

↳ Is x a free variable?

No.

↳ It is a preposition

$x = 2$

↳ predicate

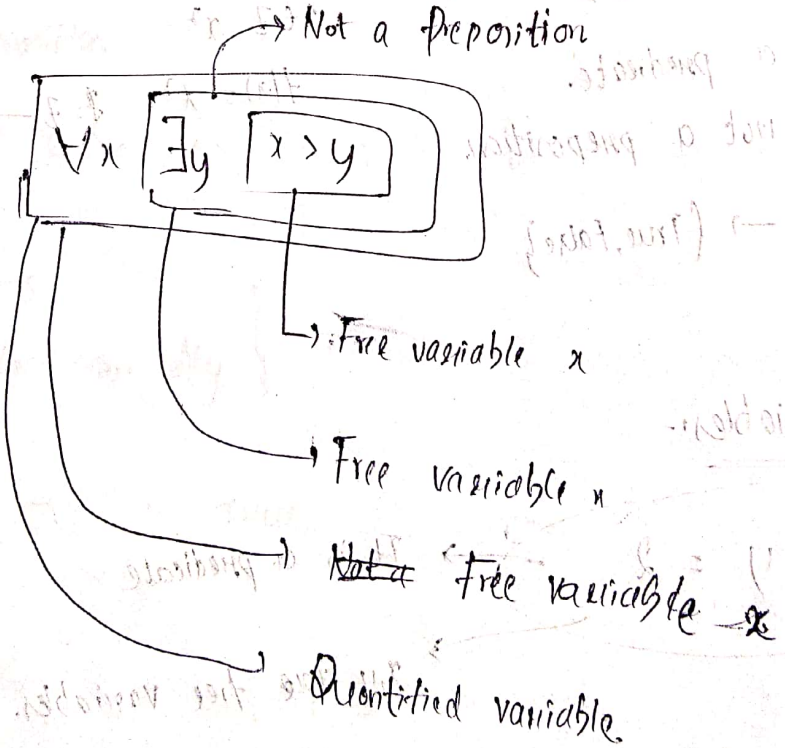
↳ but not preposition

Quantified variable $\Rightarrow x$

Q)

$\forall x \exists y : \quad x > y.$

↳ It is a preposition.



If there is a ~~Preposi~~ free variable it is not preposition.

$Even = \{x \mid \underbrace{x \bmod 2 = 0 \ \wedge \ x \in Z}\}$

↓

It is a predicate

↓

Countably infinite

$Prime = \{x \mid \forall y \ \ 1 < y \leq \sqrt{x} \ \ x \bmod y \neq 0 \wedge x, y \in Z\}$

↓

Infinite

→ Any set can be written using a predicate.

→ Set that doesn't have a predicate.

→ Fibanocci Sequence / Virahankasequence

$\{0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots\}$

Can it be represented in builder form.?

$x_n = x_{n-1} + x_{n-2}$

$x_{n-1} = x_{n-2} + x_{n-3}$

$x_{n-2} = x_{n-3} + x_{n-4}$

$x_2 = x_1 + x_0$

$\left(\frac{\sqrt{5}+1}{2}\right)^n + \left(\frac{\sqrt{5}-1}{2}\right)^{-n} = x_n$

$x_n + \ldots + x_2 = x_{n-1} \ldots + x_1 + x_{n-2} \ldots + x_0$

$Vir_0(x) \equiv x = 0$

$Vir_1(x) \equiv x < 1$

$Vir_i(x) \equiv \{\exists y, \exists z \;.\; Vir_{i-1}(y) + Vir_{i-2}(z) = Vir_i(x) \wedge y + z = x\}$

$Vir = \bigcup_{i=} Vir_i$

For, int modulo arithematic is not defined.

unsigned int modulo arithematic is defined.

x = int_max

$x+1 = x$ then x is undefined.

x = unsigned_int_max

$x = x + 1$ then x = unsigned_int_min = 0.

Every set can be represented as a predicate

$P \cup Q$

$P(x) = \{x\}$ $\}$ $P(x) \vee Q(x)$

$Q(x) = \{y\}$ $\}$

$P \cap Q$

$P(x) \wedge Q(x)$

$P \subseteq Q$ — $P(x) \rightarrow Q(x)$

~~$P(x) \rightarrow Q(x)$~~

We need to do "Breadth for Search"

$I(x)$ → Intial state

$G = (V, E)$

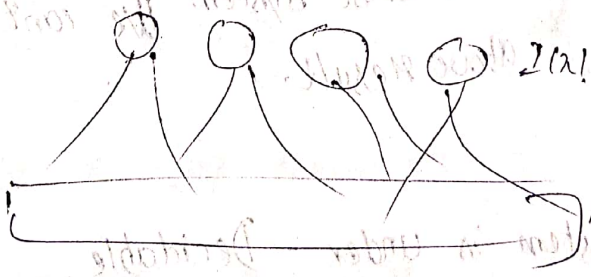
$T(x, x')$ → Transition relation / Describes all edges.

- True then there in an edge
- False then there in no edge.

⟹ Graph in described using predicates.

Q) Use predicate logic operators

$\exists x.\ I(x) \wedge T(x, x')$ = Next step of intial state (x) → Equation - ①



$x = 0$

$x := x + 2$

$x' = x + 2$ → Example of a predicate that describes my edges.

→ Solution of for Equation - ① = {2}

↳ Singleton set.

$F_0(x) = I(x)$

$F_1(x) = I(x) \vee (\exists x \cdot I(x) \wedge T(x, x'))$

$F_i(x) = F_{i-1}(x) \vee (\exists x \cdot F_{i-1}(x) \wedge T(x, x'))$

Aim :-

To check whether we reached a bad/good state.

$B(x)$ → predicate for bad state

$\exists x$
$F_i(x) \wedge B(x)$ → We can know the path to bad state

Unclecidable — There exist system We can't find the above result.

Assuming the system in under - Decidable, . . .
There are solvers

If domain is empty

$\exists x \; p(x)$ → False

$\forall x \; p(x)$ → True → $\forall x \in D \rightarrow p(x)$

$\exists x, x' \quad T(x, x') \wedge F_i(x) \wedge \neg F_i(x')$

$\exists x \quad F_{i+1}(x) \wedge \neg F_i(x)$ ⟶ If "0" stop
"1" continue.

$F_{i+1}(x) \subset F_i(x)$

$\Rightarrow$

$f(x) \equiv x^2$

$f(v) = v$ ⟶ then v is fixed point of $f(x)$

So, 0 & 1 are fixed points of $f(x) = x^2$

Function from State ① to State ②

Q) $X = 0$

$X' = X + 2$

$X = 1$ → Is my Bad state.

If Infinite state, it will not terminate

Forward Reachability Analysis" → Intial state to Bad state

Q) If Finite State, it will terminate

"Backward Reachability Analysis"

Intial State ⟵ to Bad State

Correctness

```
int max (int a, int b)
if (a > b)
return a
else
return b
```

Ans. $(result = a) \vee (result = b) \wedge (result \geq a) \wedge (result \geq b)$ (① and ②)

Predicate

Why is ② needed?

Let $a = 3, b = 5$. So, ① is true if result = 3 (③)

Do, inorder to get ⑤ result = 5, ② is needed.

③ is called Post-condition.

Precondition

Eg:- For Factorial

$n > 0 \; \& \; n < 46$

```
int fact (int n)
{ if (n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

Hoare Triple $\{P\}\ Q\ \{R\}$

↑ Precondition ← Post-condition.

Valid if Q starts with a state that satisfies "P" and it terminates in a state that satisfies "R"

$a \rightarrow b$

a = false then $a \rightarrow b \equiv$ true.

⇒ If Q is non-terminating "R" can be any-thing.

$\{P_1\}\ Q_1\ \{R_1\}$, $\{P_2\}\ Q_2\ \{R_2\}$, $R_1 \rightarrow P_2$ ← Valid ∈

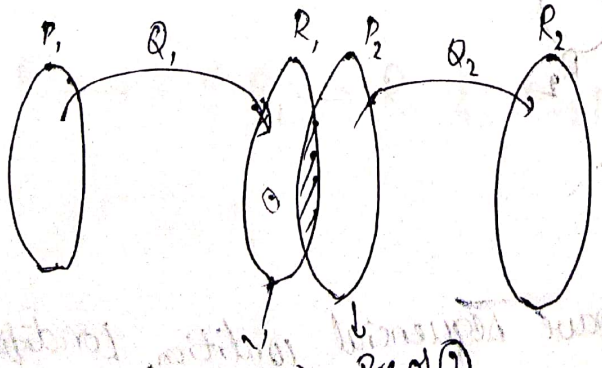
$\{P_1\}\ Q_1 ; Q_2\ \{R_2\}$ ← then

$\{P_1\}\ Q_1\ \{R_1\}$

$R_1 \rightarrow P_2$ → very important

$\{P_2\}\ Q_2\ \{R_2\}$

Sequencial Composition → $\{P_1\}\ Q_1 ; Q_2\ \{Q_2\}$



Post of ① Pre of ②

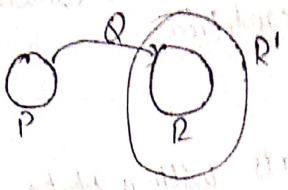
Sequencial

So, Post ① → Pre ②

$\{P\}\,Q\,\{R\}$ $\quad R \rightarrow R'$

$\{P\}\,Q\,\{R'\}$

We can always weaken/increase of Post-condition size

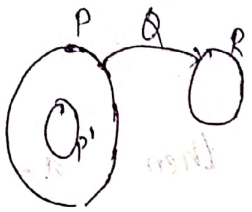


$P' \rightarrow P$

$\{P\}\,Q\,\{R\}$

$\{P'\}\,Q\,\{R\}$

We can strength/decrease of Pre-condition size

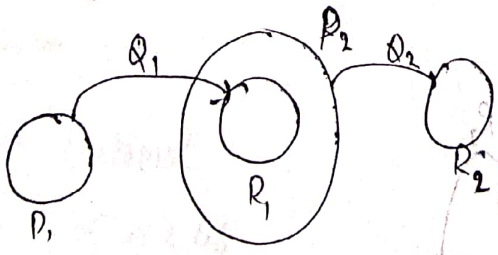


$\{False\}\,Q\,\{R\}$ → Strongest Pre-condition

$\{P\}\,Q\,\{True\}$ → Always True.

→ Weakest Post-condition.

Generally weakening - Post condition and strengthening - Pre cond should not be done.

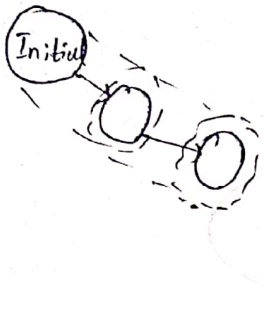


↓

In order to have sequencial condition. Condition $R_1 \rightarrow R_2$ should be done.

So, it is better to strength - post and weaken - pre.

Breadth For Search using Predicates



Predicate describing Transition states might be very large

"- - - -" indicates zone of where Transition state exist

We use this so that we can get shorter description

" " doesn't intersect with bad-state that smaller doesn't.

Q) $\dfrac{\{P_1\}S_1\{R_1\} \quad \{P_2\}S_2\{R_2\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2\ \{R\}}$

$R_1 \longrightarrow R \;\&\; R_2 \longrightarrow R \;\&\; P \rightarrow P_1 \quad P$

$P \wedge b \longrightarrow P_1$

$P \wedge \neg b \longrightarrow P_2$

Loops

$P \wedge \neg b \rightarrow R$ $\quad$ $P \wedge b \rightarrow P'$ $\quad$ $R' \wedge \neg b \rightarrow R$

$\{P \wedge b\}\ S\ \{R\}$ $\quad$ $\{P'\}\ S\ \{R'\}$ $\quad$ $R' \wedge b \rightarrow P'$

$\{P\}$ while b do S od $\{R\}$

$P \wedge b \rightarrow b$

$P' \wedge b \rightarrow$

Invariant:-

$P \rightarrow Inv$ $\quad$ $Inv \wedge b \rightarrow P'$ $\quad$ $R' \rightarrow Inv$

$Inv \wedge \neg b \rightarrow R$

```
x := 10
while (x < 10)
    x++;
assert (x ≤ 10);
```

```
x := 0
y := 0
while {x = y ∧ x ≤ 500} (x < 500)
    {
    ++x;
    ++y;        ⟶  {x = y+1 ∧ x ≤ 500}
    }
```

$x := e \{P\}$ ⟶ Pre-condition $= \{P[e/x]\}$

$x := x + 5 \{x > 10\}$

So, precondition is $\{x \geq 5\}$

$y := e \{x > 10\}$

$e = x + 5$

$x + 5 > 10$

$x > 5$

```
if (a > b)
    result a;        ——— a ≥ b
                      (A)
else
    result b;        ——— b ≥ a
                      (A)
```

$(result = a \vee result = b) \wedge (result \geq a \wedge result \geq b)$

(A)

Total Correctness

$[P]\ Q\ [R]$

⇒ If Q is executed in P, Q will terminate to R

```
while (b)
{
   // code
}
```

How do we prove a program terminates

$[P] \longrightarrow P'$

$P' \wedge b \longrightarrow Q'$

$Q' \wedge \neg b \longrightarrow R$

$R' \wedge b \longrightarrow P'$

Relations

A B C D . . .

$R \subseteq A \times B \times C \times D \ldots$

$R \subseteq A \times A$

Reflexive $\forall a \in A$ $(a, a) \in R$

Symmetric $\forall a, b \in A$ $(a,b) \in R$ then $(b,a) \in R$

Transitive $\forall a, b, c \in A$ $(a,b), (b,c) \in R$ then $(a,c) \in R$

Asymmetric.

$\exists a, b$ / such that $(a,b) \in R$ and $(b,a) \notin R$

Anti-Symmetric

If $(a, b) \in R$ and $(b, a) \in R$ then $a = b$

Total order is partial order with a condition that all elements are comparable.

Well order

Binary relation on set such that non-empty, we have a minimum.

Preposition

All Well-Orders are Total-Orders — True

Every Total-order in a well-order — False.

↳ Counter Example

$(0, 1)$ → We cannot find minimum element.

Function such that $f(v) < V$ V → local code variables

```
while(x<10)
{
    f(v)=10-x;
    ++x;
}
```

$x \leq 10$

$Inv \wedge (x < 10)$

$f(v) > 0)$

$f(v) = 0$

↳ Evalution of function

```
i := 0
max := A[i]
while ( i < length)
{
    if ( A[i] > max)
        max = A[i];
    i++;
}
```

Precondition

length > 0

Post condition

$\exists i$ result = A[i] $\wedge$ $\forall j$ result $\geq$ A[j]

$0 \leq j < length$

Remove to prove.

result = max $\longleftarrow$ $\forall j$ $0 \leq j < length$

max $\geq$ A[j]

Invariant

$i \leq length \wedge 0 \leq k < i \Longrightarrow max \geq A[k]$

Invariant + $\sim$ loop condition $\longrightarrow$ Post condition of loop

while(i < length)

{ $\longrightarrow$ Inv $\wedge$ i < length

if(A[i] > max)

max = A[i]; $\longrightarrow$ Inv $\wedge$ i < length $\wedge$ A[i] $\leq$ max

$\longrightarrow$ Inv $\wedge$ $i+1 \leq length \wedge 0 \leq k < i+1 \Rightarrow max \geq A[k]$

i++;

$\longrightarrow$ Invariant Holds

}

Inv $\wedge$ i<length $\wedge$ A[i] $\leq$ max

$i+1 \leq$ length $\wedge$ $0 \leq k < i+1$ $\Rightarrow$ max $\geq$ A[k] — (1)

[] Equal.

$\Rightarrow$ i<length $\wedge$ $0 \leq k < i+1$ $\wedge$ A[i] $\leq$ (max) $\Rightarrow$ max $\geq$ A[k] — (2)

This is if ~~and~~ if-loop is not executed.

if "if-loop" is executed,

Inv $\wedge$ i<length $\wedge$ A[i] > max. $\Rightarrow$ max $\geq$ A[k]

$\Rightarrow$ i<length $\wedge$ $0 \leq k < i$ $\wedge$ A[i] > max $\Rightarrow$ max $\geq$ A[k]

$\Rightarrow$ i<length $\wedge$ $0 \leq k < i$ $\Rightarrow$ max $\geq$ A[k]. — (3)

(1) $\neq$ (3)

(1) = (2)

Quantifier invariant.