Aman Choudhary SR. No: 17920

ASSIGNMENT-1

(1)

Let us consider the set of real numbers between 0 and 1,

$$A = [0,1] \subseteq R$$

We define a partial order on A using $\leqslant$,

$$P = (A, \leqslant)$$

$\leqslant$ is reflexive as $x \leqslant x$, $\forall\ x \in A$

$\leqslant$ is anti-symmetric as $(x \leqslant y$ AND $y \leqslant x) \Rightarrow x = y$, $\forall\ x, y \in A$

$\leqslant$ is transitive as $(x \leqslant y$ AND $y \leqslant z) \Rightarrow x \leqslant z$, $\forall x, y, z \in A$

Now,

P is a lattice because $\forall\ x, y \in A$,

$$x \sqcup y = \max(x, y)$$
$$x \sqcap y = \min(x, y)$$

Also,

0 is the least element for P, as $\forall x \in A$, $0 \leqslant x$

1 is the greatest element for P, as $\forall x \in A$, $x \leqslant 1$

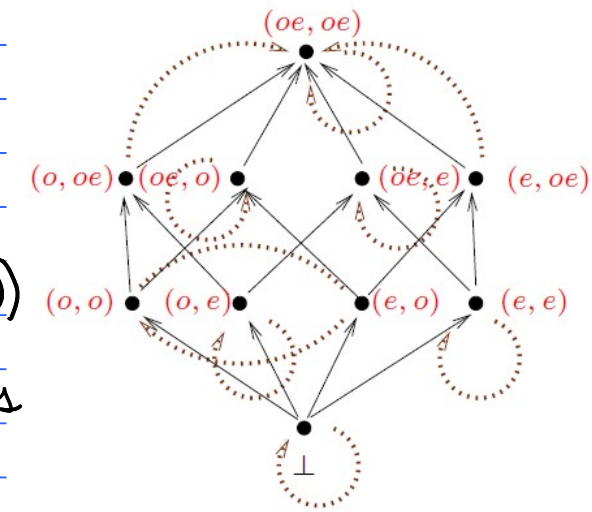
Let us consider the interval,

$$B = [0, 1) \subset A$$

If we assume, $l \in B$ to be the lub of B, then we can always show $l' \in B$ such that $l \leqslant l'$. Hence, there exists no lub for $B \subset A$. Hence, P is not a complete lattice.

Hence, we cannot say that a lattice will be complete, if there exists both a least and a greatest element.

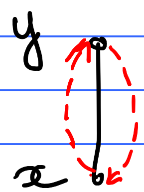
(2)



(a) Yes, the lattice is monotonic, because,

$$\forall x \, \forall y \, ((x \leqslant y) \Rightarrow (f(x) \leqslant f(y))$$

Also, there are no mapping arrows that are going downwards. Hence, there can be no loops of the form shown below which can violate monotonicity.



Here, $x \leqslant y$
$f(x) \geqslant f(y)$

(c) No, the lattice is not distributive.

Take, $d_1 = (o,o)$ and $d_2 = (e,e)$

$f(d_1) = (e,o)$
$f(d_2) = (e,e)$
$f(d_1) \sqcup f(d_2) = (e,oe)$ — (1)

$d_1 \sqcup d_2 = (oe,oe)$
$f(d_1 \sqcup d_2) = (oe,oe)$ — (2)

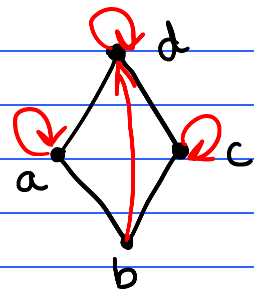
From (1) & (2),
$$f(d_1) \sqcup f(d_2) \neq f(d_1 \sqcup d_2)$$

(b) Yes, the lattice is continuous because,

For any ascending chain in the lattice,
$$f(\sqcup X) = \sqcup f(X)$$

③

(a) Counter example to show every increasing function is not monotonic:



$f(a) = a \geq a$ $\quad$ $f(c) = c \geq c$

$f(b) = d \geq b$ $\quad$ $f(d) = d \geq d$

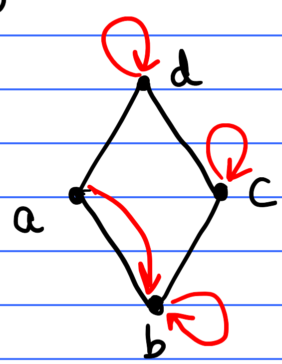
$\therefore f$ is increasing.

Now,

$$b \leq a$$

But, $f(b) = d \geq f(a) = a$

(b) Counter example to show every monotic function is not increasing:



$b \leq a$ and $f(b) = b \leq f(a) = b$

$b \leq c$ and $f(b) = b \leq f(c) = c$

$b \leq d$ and $f(b) = b \leq f(d) = d$

$a \leq d$ and $f(a) = b \leq f(d) = d$

$c \leq d$ and $f(c) = c \leq f(d) = d$

$\therefore f$ is monotonic

Now,

$$a \geq f(a) = b$$

(4)

(a) NOTATION:

Var $\rightarrow$ Set of variables in the program
Val $\rightarrow$ Set of values that the variables can assume
Num $\rightarrow$ Set of statement numbers in the program

State is defined as Var $\rightarrow$ Val
VDef is defined as Var $\rightarrow$ Num

ExtState = State $\times$ VDef

- Concrete Transfer functions are defined as,

$$nstateExt_{MN} : ExtState \rightarrow 2^{ExtState}$$

$$nstateExt'_{MN} : 2^{ExtState} \rightarrow 2^{ExtState}$$

$$nstateExt'_{MN}(E \in 2^{ExtState}) = \bigcup_{e \in E} nstateExt_{MN}(e)$$

- Collecting semantics is defined as,

$$CS : ProgramPoints \rightarrow 2^{ExtState}$$

$$CS(t \in ProgramPoints) = \bigcup_{\substack{p \text{ path from} \\ I \text{ to } t}} nstateExt'_{p}(E_0),$$

where E_0 is the set of initial concrete states.

- Concrete Transfer function for assignment statement

Let us consider an assignment statement S_{MN} between program points M and N, whose LHS is variable $v_i \in Var$

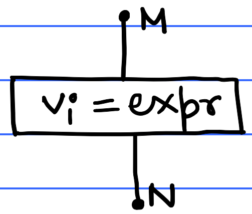


Fig 1.

Let $e_M \in ExtState$ be a concrete state at point M. We need to calculate $E_N \in 2^{ExtState}$ for point N. For that we will define the concrete transfer function such that,

$$nstateExt\_ASSIGN_{MN}(e_M) = E_N$$

Now, we will use the 'nstate' transfer function to transfer the 'State' component of e_M.

$$nstate : State \rightarrow 2^{State}$$

So,

$$nstate_{MN}(e_M.State) = E_{N\_State}$$

Next, we need to transfer the 'VDef' component of e_M.

Let, E_{N_VDef} be a singleton set such that,

$$E_{N\_VDef} = \left\{ x \;\middle|\; \begin{array}{l} x \in VDef, \text{ and} \\ x[v_i] = stmtNum(S_{MN}) \text{ and} \\ \forall v_j \neq v_i,\; x[v_j] = e_M.VDef[v_j] \end{array} \right\}$$

Now, we can define nstateExt_ASSIGN as

$$nstateExt\_ASSIGN_{MN}(e_M) = E_{N\_State} \times E_{N\_VDef} = E_N$$

- Concrete Transfer function for conditional statement

Let us consider a conditional node s_M.

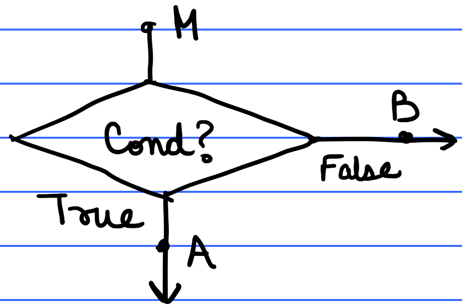


Fig 2

Let $e_M \in ExtState$ be a concrete state at point M. We need to calculate E_A and E_B such that $E_A, E_B \in 2^{ExtState}$ for points A and B. We define the transfer function as follows:

$$nstateExt\_COND_{MA}(e_M) = E_A \quad \text{, for TRUE branch}$$
$$nstateExt\_COND_{MB}(e_M) = E_B \quad \text{, for FALSE branch}$$

If condition at node s_M evaluates to TRUE,

$$nstateExt\_COND_{MA}(e_M) = \{e_M\} = E_A$$
$$nstateExt\_COND_{MB}(e_M) = \varphi = E_B$$

If condition at node s_M evaluates to FALSE,

$$nstateExt\_COND_{MA}(e_M) = \varphi = E_A$$
$$nstateExt\_COND_{MB}(e_M) = \{e_M\} = E_B$$

- Identifying the set of definitions that reach 't' using the collecting semantics at program point 't'

Let $CS(t) \in 2^{ExtState}$ be the collecting semantics at 't',
Let D_{v_i} be the set of definitions for variable $v_i \in Var$, that reaches 't'. Then,

$$D_{v_i} = \bigcup_{e \in CS(t)} e.VDef[v_i]$$

To get the set of all definition, $D = \bigcup_{v \in Var} D_v$

(b) Now, $D = 2^{Num}$. We need to define transfer functions $f_n : D \to D$

- Transfer function for assignment state (refer Fig 1)

  Let $d_M \in D$ and $d_N \in D$ be the abstract states at points M and N respectively. Then,

$$f_{MN}(d_M) = d_M - S_{v_i} \cup \{stmtNum(S_{MN})\} = d_N$$

Here, $S_{v_i} \in D$ is the set of statements whose definition for variable v_i reaches M.

- Transfer function for condition node (refer Fig 2)

  Let $d_M, d_A, d_B \in D$ be the abstract states at points M, A and B respectively. Then,

  If condition at node S_M evaluates to TRUE,

$$f_{MA}(d_M) = d_M = d_A$$
$$f_{MB}(d_M) = \phi = d_B$$

  If condition at node S_M evaluates to FALSE,

$$f_{MA}(d_M) = \phi = d_A$$
$$f_{MB}(d_M) = d_M = d_B$$

- For the abstract lattice D, the order would be defined by $\subseteq$ i.e for $d_1, d_2 \in D$, $d_1 \leq d_2$ if $d_1 \subseteq d_2$.
  Also, the join operation would be union such that

$$d_1 \sqcup d_2 = d_1 \cup d_2$$

(c) $\gamma_D : D \longrightarrow 2^{ExtState}$

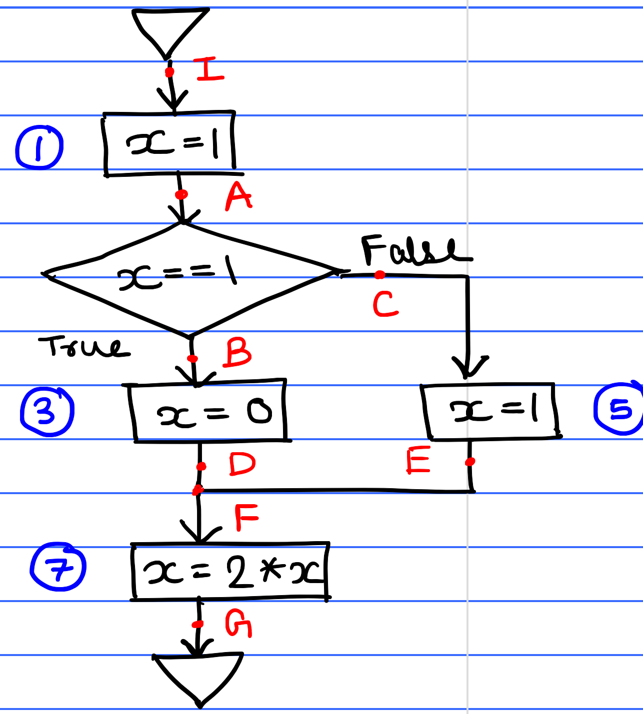
Let $d \in D$ be an abstract state at point 't'.

$$\gamma_D(d) = \left\{ e \;\middle|\; \begin{array}{c} \exists e \in ExtState, \exists v \in Var, \exists s \in Num, \\ e.VDef[v] = s \end{array} \right\}$$

The output of the above function is a set of concrete states which will over-approximate the collecting semantics at that point.

(d) Sample Program:

```
1  x = 1
2  if (x == 1) {
3      x = 0;
4  } else {
5      x = 1;
6  }
7  x = 2 * x
```



Abstract States:

I →	{ } .		{ }
A →	{1}		{1}
B →	{1}		{1}
C →	{1}		{ }
D →	{3}		{3}
E →	{5}	⊃	{ }
F →	{3,5}	⊃	{3}
G →	{7,5}	⊃	{7}

Actual definitions that can reach program points

For E, F & G, the abstract states are a strict superset of the actual definitions that reach there.

(e) If we change the join operation to <u>intersection</u>, then in the abstract JOP, we will only have those definitions which reach a program point over all paths possible from I. This will be an under-approxiation of the set of actual definitions, because the join will not include those definition that don't reach a point through each possible path.