Exercise 1

$\forall T.\ T \times List[T] \Rightarrow List[T]$

```
def CONS[T](x:T, lst:List[T]):List[T]={...}
def listInt() : List[Int] = {...}
def listBool() : List[Bool] = {...}
```

:TA :TB TD

```
def baz(a, b) = CONS(a(b), b)
def test(f,g) =
      (baz(f,listInt), baz(g,listBool))
```

TC. TB.

$TA = (TB \Rightarrow TC)$

$T_1 \times List[T_1] \Rightarrow List[T_1]$

$= TC \times TB \Rightarrow TD$

$TB = List[T_1] \quad TD = List[T_1]$

$TC = T_1$

Solving 'baz'

def CONS[T](x:T, lst:List[T]) : List[T] = {...}

def baz(a:TA, b:TB):TD = CONS(a(b):TC, b:TB):TD

$TA = (TB \Rightarrow TC)$

$CONS : T_1 \times List[T_1] \Rightarrow List[T_1]$

$TC = T_1$

$TB = List[T_1]$

$TD = List[T_1]$

$TA = (List[T_1] \Rightarrow T_1)$

Solved form. Generalize over T_1

def baz[T1](a: List[T1]=>T1,b:List[T1]):List[T1] =

CONS[T1](a(b),b)

Using generalized ‘baz’

def baz[T1](a: List[T1]=>T1,b:List[T1]):List[T1] =
        CONS[T1](a(b),b)

def test(f,g) = (baz(f,listInt), baz(g,listBool))

Example 2:

```
def baz(a, b) = a(b) :: b
```

The operator :: concatenates a list (type List[A]) with an element of the appropriate type A.

Example 3:

```
def twice(f) = (x) => f(f(x))
def succ(x) = x + 1
twice(succ)(5)
```

Example 4

def selfApp(f) = f(f)

Physical Units Type Inference: 1

```
def coulomb(k, q1, q2,r) = {
   (k* q1 * q2)/(r*r)
}
```

Physical Units Type Inference: 2

```
def energy(m,g,h,v) = {
  m*g*h + m*v*v/2
}
```

Data-Flow Analysis

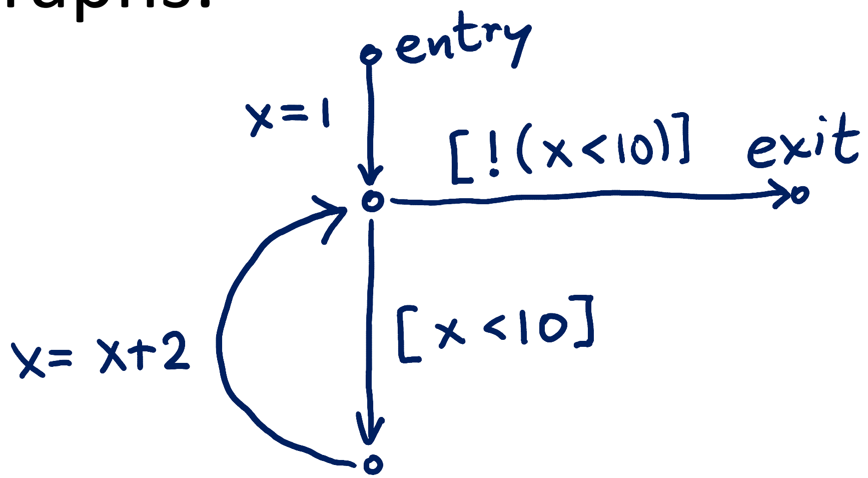
Goal of Data-Flow Analysis

Automatically compute information about the program

- Use it to report errors to user (like type errors)
- Use it to optimize the program

Works on control-flow graphs:

```
x = 1
while (x < 10) {
  x = x + 2
}
```



How We Define It

- Abstract Domain **D** (Data-Flow Facts): which information to compute?
  - **Example**: interval for each variable x:[a,b], y:[a’,b’]
- Transfer Functions [[**st**]] for each statement **st**, how this statement affects the facts
  - Example:

$$[\![x = x+2]\!](x:[a,b], \ldots) = (x:[a+2, b+2], \ldots)$$

$x:[a,b] \quad y:[c,d]$

$x = x+2$

$x:[a+2, b+2], \ y:[c,d]$

Find Transfer Function: Plus

Suppose we have only two integer variables: x,y

$\circ$ x: [a,b] y: [c,d]

$\downarrow$ x= x+y

$\circ$ x: [a',b'] y: [c',d']

If $a \leq x \leq b \qquad c \leq y \leq d$

and we execute **x= x+y**

then $x' = x+y$

$y' = y$

so

$\leq x' \leq$

$\leq y' \leq$

So we can let

a'= a+c b' = b+d

c'=c d' = d

Find Transfer Function: Minus

Suppose we have only two integer variables: x,y

$x : [a,b] \quad y : [c,d]$

$y = x - y$

$x : [a',b'] \quad y : [c',d']$

If

and we execute **y= x-y**

then

So we can let

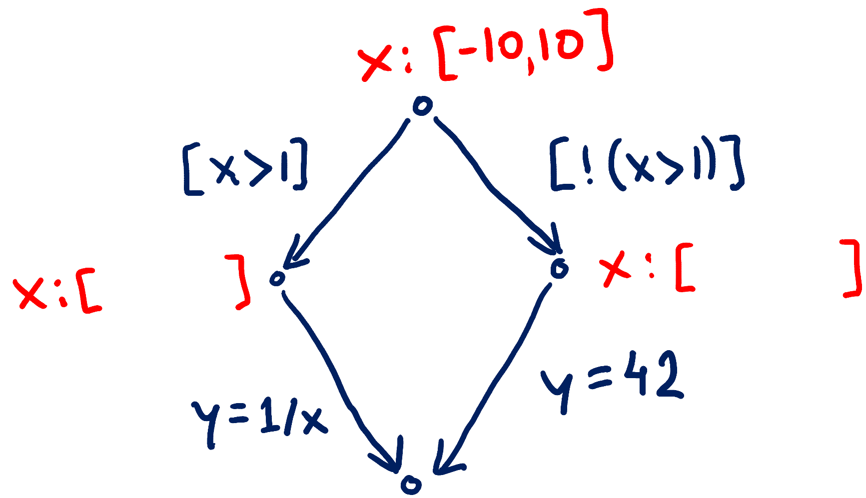
$a' = a \qquad b' = b$

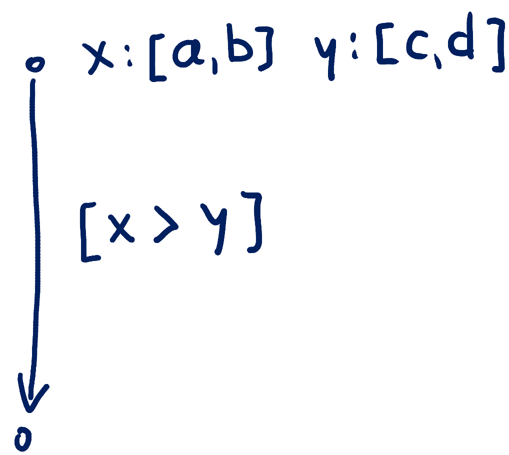
$c' = a - d \qquad d' = b - c$

Transfer Functions for Tests

x : [-10,10]

```
if (x > 1) {
  x :
  y = 1 / x
} else {
  x :
  y = 42
}
```





Merging Data-Flow Facts

x: [-10,10]   y: [-1000,1000]

if (x > 0) {

  x: y:

  y = x + 100

  x: y:

} else {

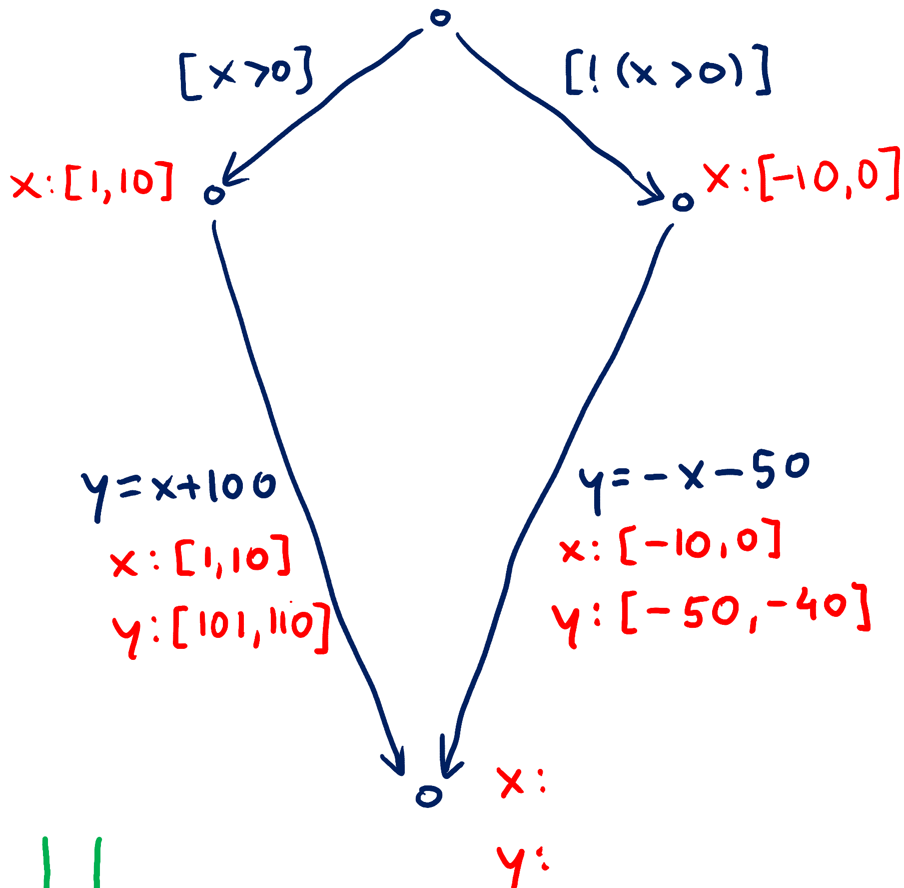
  x: y:

  y = -x – 50

  x: y:

}

x: y:



join ⊔

$[a,b] \sqcup [c,d] = [\min(a,c), \max(b,d)]$

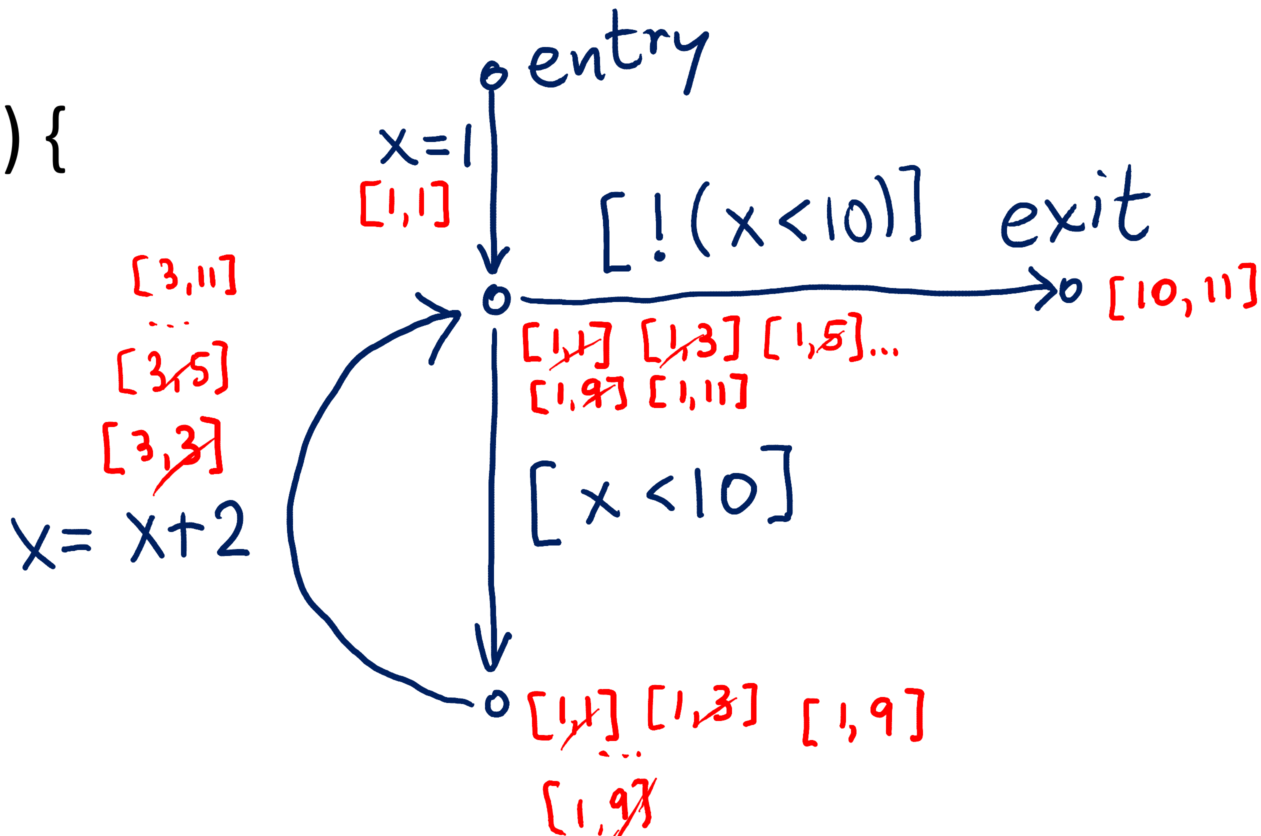
Handling Loops: Iterate Until Stabilizes

$[1,1] \sqcup [3,3] = [1,3]$

$[1,1] \sqcup [3,5] = [1,5]$

Compiler learned some facts! ☺

```
x = 1
```
$x \in [1,1]$
```
while (x < 10) {
```
$x \in [1,9]$
```
  x = x + 2
```
$x \in [3,11]$
```
}
```
$x \in [10,11]$



Data-Flow Analysis Algorithm

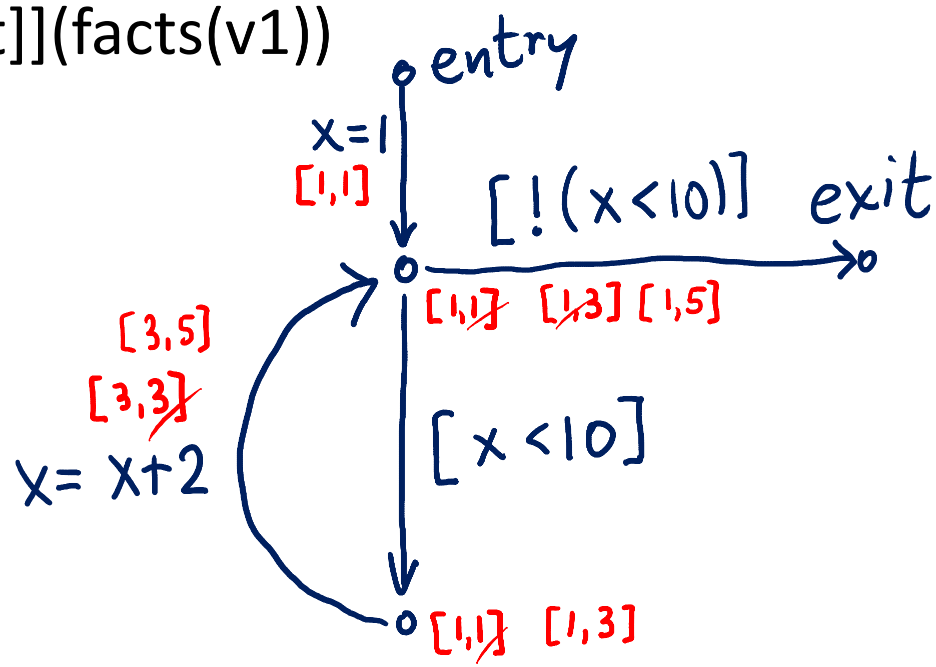
var facts : Map[Vertex,Domain] = Map.withDefault(empty)
facts(entry) = initialValues *// change*

while (there was change)
  pick edge (v1,statmt,v2) from CFG
        such that facts(v1) was changed
  facts(v2)=facts(v2) **join** [[statmt]](facts(v1))
}

$[1,1] \sqcup [3,3] = [1,3]$

$[1,1] \sqcup [3,5] = [1,5]$

Order does not matter for the end result, as long as we do not permanently neglect any edge whose source was changed.



Handling Loops: Iterate **Until Stabilizes**

Compiler learns
some facts, but only after long time

```
x = 1
n = 100000
while (x < n) {
  x = x + 2
}
```

Handling Loops: Iterate **Until Stabilizes**

For unknown program inputs it may be practically impossible to know how long it takes

```
var x : BigInt = 1
var n : BigInt = readInput()
while (x < n) {
  x = x + 2
}
```

Solutions

- smaller domain, e.g. only certain intervals $[a,b]$ where a,b in $\{-\infty,-127,-1,0,1,127,\infty\}$
- *widening* techniques (make it less precise on demand)

Size of analysis domain

Interval analysis:

$D_1 = \{ [a,b] \mid a \leq b,\ a,b \in \{-M,-127,-1,0,1,127,M-1\}\} \cup \{\perp\}$

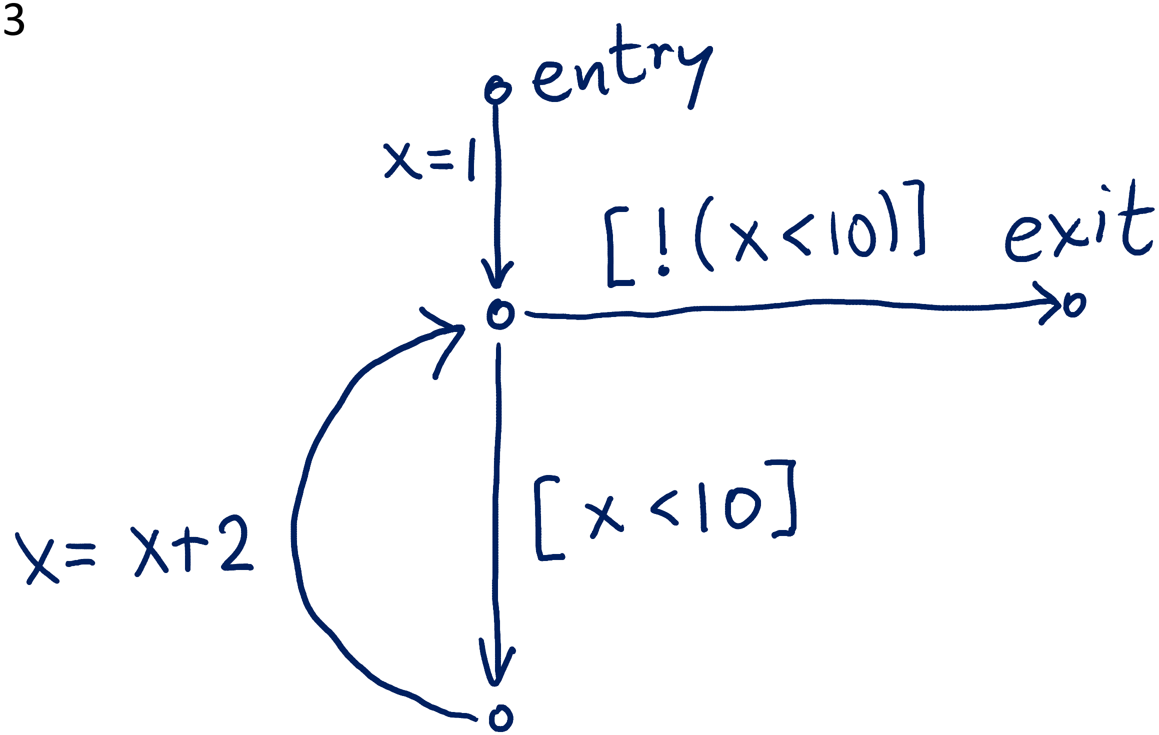
Constant propagation:

$D_2 = \{ [a,a] \mid a \in \{-M,-(M-1),\ldots,-2,-1,0,1,2,3,\ldots,M-1\}\} \cup \{\perp\}$

suppose M is 2^{63}

$|D_1| =$

$|D_2| =$



How many steps does the analysis take to finish (converge)?

Interval analysis:

D_1 = { [a,b] | a ≤ b, a,b∈{-M,-127,-1,0,1,127,M-1}} U {⊥}

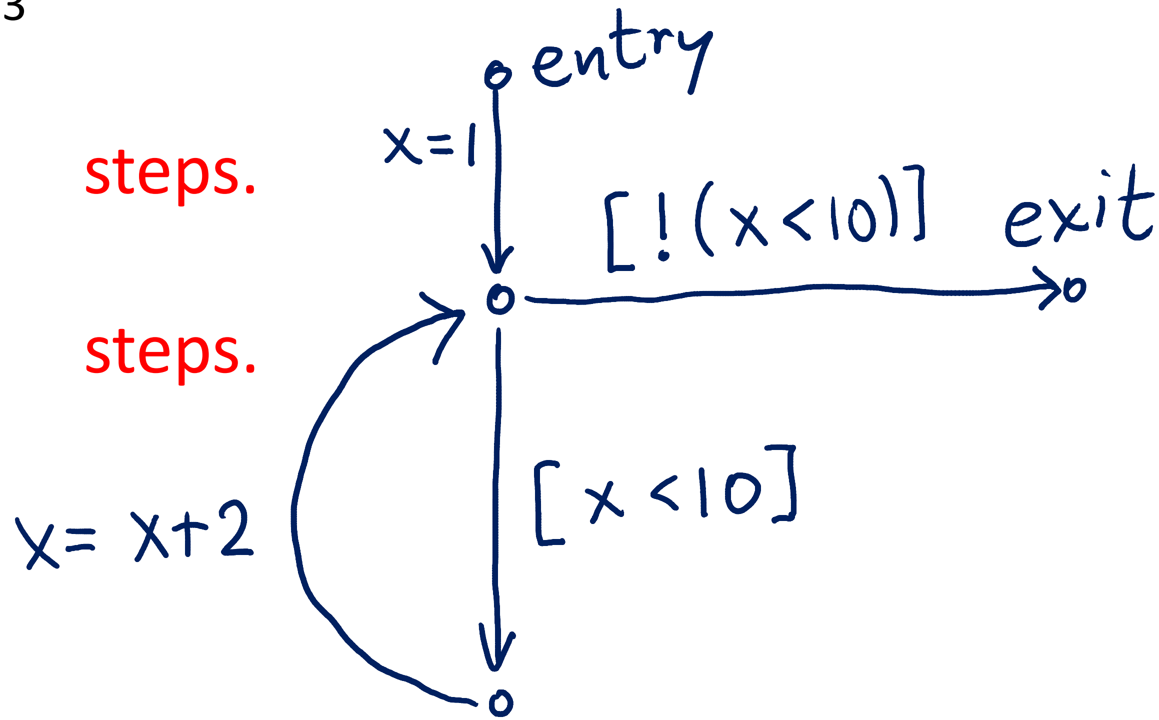
Constant propagation:

D_2 = { [a,a] | a∈{-M,-(M-1),…,-2,-1,0,1,2,3,…,M-1}} U {⊥}

suppose M is 2^{63}

With D_1 takes at most steps.

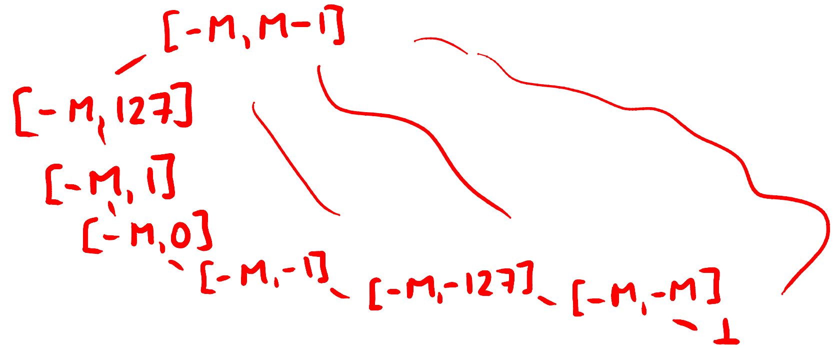
With D_2 takes at most steps.



Termination Given by Length of Chains

Interval analysis:

$D_1 = \{ [a,b] \mid a \le b,\ a,b \in \{-M,-127,-1,0,1,127,M-1\}\} \cup \{\perp\}$

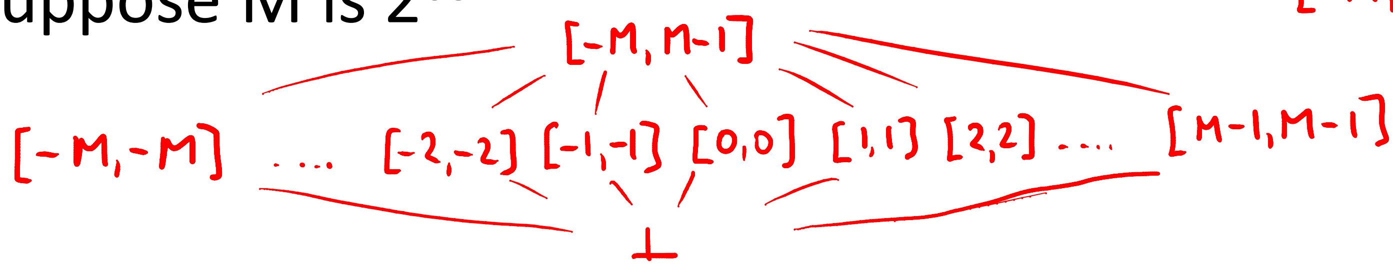


Constant propagation:

$D_2 = \{ [a,a] \mid a \in \{-M,\ldots,-2,-1,0,1,2,3,\ldots,M-1\}\} \cup \{\perp\} \cup \{T\}$

[-M, M-1]

suppose M is 2^{63}



Domain is a **lattice**. Maximal chain length = **lattice height**

Lattice for intervals [a,b] where $a,b \in \{-M,-127,0,127,M-1\}$