Abstract Interpretation
Lattice

**Partial order:** binary relation \( \leq \) (subset of some \( D^2 \)) which is

- reflexive: \( x \leq x \)
- anti-symmetric: \( x \leq y \land y \leq x \rightarrow x = y \)
- transitive: \( x \leq y \land y \leq z \rightarrow x \leq z \)

**Lattice** is a partial order in which every two-element set has **least among its upper bounds** and **greatest among its lower bounds**

- Lemma: if \((D, \leq)\) is lattice and \(D\) is finite, then lub and glb exist for every finite set

\[ \cap \cup \sqcup \{a, b, c\} \]
Graphs and Partial Orders

• If the domain is finite, then partial order can be represented by directed graphs
  – if \( x \leq y \) then draw edge from \( x \) to \( y \)

• For partial order, no need to draw \( x \leq z \) if \( x \leq y \) and \( y \leq z \). So we only draw non-transitive edges

• Also, because always \( x \leq x \), we do not draw those self loops

• Note that the resulting graph is acyclic: if we had a cycle, the elements must to be equal
Domain of Intervals $[a,b]$ where $a,b \in \{-M,-127,0,127,M-1\}$
Defining Abstract Interpretation

Abstract Domain $D$ describing which information to compute – this is often a lattice

- inferred types for each variable: $x:T1$, $y:T2$
- interval for each variable $x:[a,b]$, $y:[a',b']$

Transfer Functions, $[[st]]$ for each statement $st$, how this statement affects the facts $D \rightarrow D$

- Example:

$$[[x = x + 2]] (x:[a,b], ...)
= (x:[a+2,b+2], ...)$$

$$x = x + 2$$

$$x:[a+2,b+2], y:[c,d]$$
For now, we consider arbitrary integer bounds for intervals

• Thus, we work with BigInt-s
• Often we must analyze machine integers
  – need to correctly represent (and/or warn about) overflows and underflows
  – fundamentally same approach as for unbounded integers
• For efficiency, many analysis do not consider arbitrary intervals, but only a subset of them $W$
• We consider as the domain
  – empty set (denoted $\bot$, pronounced “bottom”)
  – all intervals $[a,b]$ where $a,b$ are integers and $a \leq b$, or where we allow $a = -\infty$ and/or $b = \infty$
  – set of all integers $[-\infty, \infty]$ is denoted $T$, pronounced “top”
Find Transfer Function: Plus

Suppose we have only two integer variables: $x, y$

If $a \leq x \leq b$ and $c \leq y \leq d$ and we execute $x = x + y$

then $x' = x + y$

$y' = y$

so

$\leq x' \leq b + d$

$\leq y' \leq d$

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: $x,y$

If $y = x - y$ and we execute $y = x - y$ then

So we can let

$a' = a$

$b' = b$

$c' = a - d$

$d' = b - c$
Transfer Functions for Tests

Tests e.g. \([x>1]\) come from translating if, while into CFG

\[
\begin{align*}
\text{x} : & \ [-10,10] \\
\text{if (x > 1) \{ } & \\
\quad \text{x} : & \\
\quad \text{y} = 1 / x & \\
\text{\} else \{ } & \\
\quad \text{x} : & \\
\quad \text{y} = 42 & \\
\text{\}}
\end{align*}
\]
Joining Data-Flow Facts

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

if \(x > 0\) {
    \[y = x + 100\]
} else {
    \[y = -x - 50\]
}

\[
\text{join}
\]

\[
x: [1,10] \quad y: [-101,110]
\]

\[
x: [-10,0] \quad y: [-50,-40]
\]

\[
x: [-10,10] \quad y: [-50,110]
\]
Handling Loops: Iterate Until Stabilizes

\[ x = 1 \]

\[ \text{while} \ (x < 10) \{ \]
\[ x = x + 2 \]
\[ \} \]
Analysis Algorithm

```java
var facts : Map[Node,Domain] = Map.withDefault(empty)
facts(entry) = initialValues

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

Order does not matter for the end result, as long as we do not permanently neglect any edge whose source was changed.
```
var facts : Map[Node,Domain] = Map.withDefault(empty)
var worklist : Queue[Node] = empty

def assign(v1:Node,d:Domain) = if (facts(v1)!=d) {
    facts(v1)=d
    for (stmt,v2) <- outEdges(v1) { worklist.add(v2) }
}
assign(entry, initialValues)

while (!worklist.isEmpty) {
    var v2 = worklist.getAndRemoveFirst
    update = facts(v2)
    for (v1,stmt) <- inEdges(v2) {
        update = update join transferFun(facts(v1),stmt) }
    assign(v2, update)
}
Exercise: Run range analysis, prove that **error** is unreachable

```c
int M = 16;
int[M] a;
x := 0;
while (x < 10) {
    x := x + 3;
}
if (x >= 0) {
    if (x <= 15)
        a[x]=7;
    else
        error;
} else {
    error;
}
```

*checks array accesses*
int M = 16;
int[M] a;
x := 0;
while (x < 10) {
    x := x + 3;
} checks array accesses
    if (x >= 0) {
        if (x <= 15)
            a[x]=7;
        else
            error;
    } else {
        error;
    } else {
        error;
}
int M = 16;
int[M] a;
x := 0;
while (x < 10) {
    x := x + 3;
}  
check array accesses
if (x >= 0) {
    if (x <= 15)
        a[x]=7;
    else
        error;
} else {
    error;
}
Remove Trivial Edges, Unreachable Nodes

int M = 16;
int[M] a;
x := 0;
while (x < 10) {
    x := x + 3;
} checks array accesses
if (x >= 0) {
    if (x <= 15)
        a[x]=7;
    else error
} else {
    error;
} else {
    error;
}

Benefits:
- faster execution (no checks)
- program cannot crash with error

$
\begin{align*}
    M &\rightarrow [16,16], x \rightarrow [0, 9] \\
    x &= x + 3 \\
    M &\rightarrow [16,16], x \rightarrow [0, 12] \\
    x &\geq 10
\end{align*}
$

$
\begin{align*}
    M &\rightarrow [16,16], x \rightarrow [10, 12] \\
    a[x] &= 7
\end{align*}
$
Constant Propagation Domain

Domain values $D$ are:
- intervals $[a,a]$, denoted simply ‘$a$’
- empty set, denoted $\bot$ and set of all integers $T$

Formally, if $\mathbb{Z}$ denotes integers, then

$$D = \{ \bot, T \} \cup \{ a \mid a \in \mathbb{Z} \}$$

$D$ is an infinite set
**Constant Propagation Transfer Functions**

For each variable \(x,y,z\) and each CFG node (program point) we store: \(\perp\), a constant, or \(\top\)

![Diagram of \(x = y + z\)]

```scala
abstract class Element
case class Top extends Element
case class Bot extends Element
case class Const(v: Int) extends Element

var facts : Map[Nodes, Map[VarNames, Element]]

what executes during analysis of \(x = y + z\):

oldY = facts(v1)("y")
oldZ = facts(v1)("z")
newX = tableForPlus(oldY, oldZ)
facts(v2) = facts(v2) join facts(v1).updated("x", newX)
```

```scala
def tableForPlus(y: Element, z: Element) = (x, y) match {
  case (Const(cy), Const(cz)) => Const(cy + cz)
  case (Bot, _) => Bot
  case (_, Bot) => Bot
  case (Top, Const(cz)) => Top
  case (Const(cy), Top) => Top
}
```
Run Constant Propagation

What is the number of updates?

```plaintext
x = 1
n = 1000
while (x < n) {
    x = x + 2
}
```

```plaintext
x = 1
n = readInt()
while (x < n) {
    x = x + 2
}
```
Observe

• Range analysis with end points $W = \{-128, 0, 127\}$ has a finite domain

• Constant propagation has infinite domain (for every integer constant, one element)

• Yet, constant propagation finishes sooner!
  – it is not about the size of the domain
  – it is about the height
Height of Lattice: Length of Max. Chain

height=5
size=14

height=2
size=∞