

# Data Flow Analysis

---

## Why:

- Provide information about a program manipulates its data.
- Study function's behavior.
- To help build control flow information.
- Program understanding (a function sorts an array!).
- Generating a model of the original program and verify the model.
- Program validation.
- ...

# Reaching Definitions

---

A particular definition of a variable is said to reach a given point if

- there is an execution path from the definition to that point
- the variable might *may* have the value assigned by the definition.

In general undecidable.

# Different types of analysis

---

- Intra procedural analysis.
- Whole program (inter-procedural) analysis.
- Generate intra procedural analysis and extend it to whole program.

# Iterative Dataflow Analysis

---

- Build a collection of data flow equations.
- Solve it iteratively.
- Start from an conservative set of initial values.

# Example program

---

```
int f(int n){  
  
    int i, j;  
1.    i=0;  
2.    n=2;  
3.    if (n == 1)  
4.        i = 2;  
5.    while (n > 0){  
  
6.        j = n+1;  
7.        n = g(n,i);  
    }  
8    return j;  
}
```

# Definitions

---

**GEN** :  $GEN(b)$  returns the set of definitions generated in the basic block  $b$ ; assigned values in the block and not subsequently killed in it.

**KILL** :  $KILL(b)$  returns the set of definitions killed in the basic block  $b$ .

**IN** :  $IN(b)$  returns the set of definitions reaching the basic block  $b$ .

**OUT** :  $OUT(b)$  returns the set of definitions going out of basic block  $b$ .

**PRSV** : Negation of KILL

# Representation and Initialization

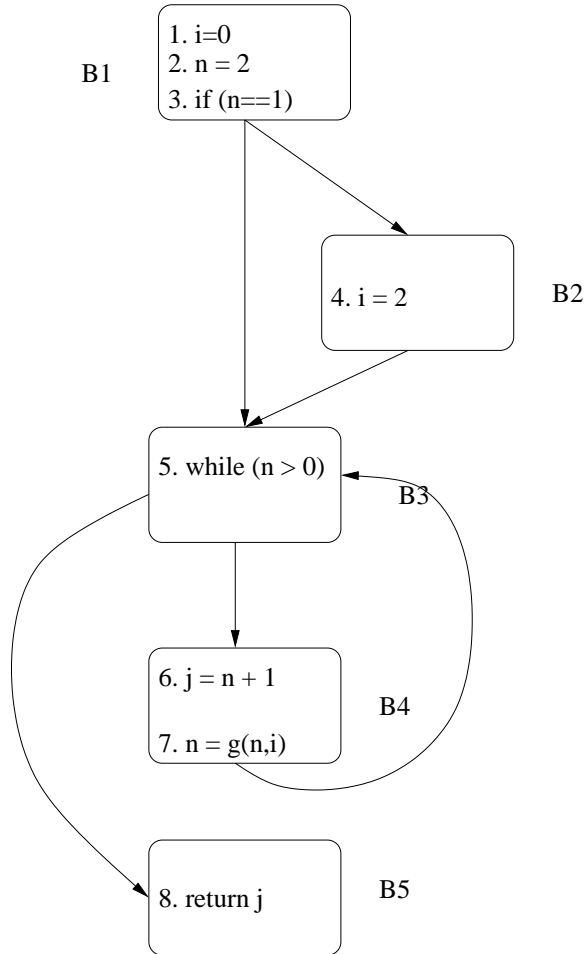
---

- Set representation. e.g.  $\text{GEN}(\text{B4}) = \{5,6\}$
- Bit Vector representation.  $\text{GEN}(\text{B4}) = \langle 0,0,0,0,1,1 \rangle$

## Initialization

- GEN, KILL and OUT are created in each basic block. But IN needs to be initialized to something safe.
- $\text{IN}(\text{entry}) = \{\}$

# Reaching definitions



$\text{Gen}(B1)$	$=$	$\{1,2\}$	$\text{Kill}(B1) = \{\}$
$\text{Gen}(B2)$	$=$	$\{4\}$	$\text{Kill}(B2) = \{1\}$
$\text{Gen}(B3)$	$=$	$\{\}$	$\text{Kill}(B3) = \{\}$
$\text{Gen}(B4)$	$=$	$\{6, 7\}$	$\text{Kill}(B4) = \{2,6,7\}$
$\text{Gen}(B5)$	$=$	$\{\}$	$\text{Kill}(B5) = \{\}$

## Computing Reaching Definitions

---

$$\text{OUT}(b) = \text{GEN}(b) \cup (\text{IN}(b) - \text{KILL}(b))$$

$$\text{IN}(b) = \bigcup_{p \in \text{pred}(b)} \text{OUT}(p)$$

# Lattice

---

**What** : Lattice is an algebraic structure.

**Why** : To represent *abstract properties* of variables, expressions, functions, etc etc.

- Values,
- Attributes,

**Why “abstract”?** Exact interpretation (execution) gives exact values, abstract interpretation gives abstract values.

## Lattice contd.

---

A lattice  $L$  consists of a set of *elements* and two n'ary operations.

**meet**( $\sqcap$ ) : Greatest lower bound.

**join**( $\sqcup$ ) : Lowest upper bound.

Each lattice has two special elements:

Bottom ( $\perp$ )

Top ( $\top$ )

# Lattice contd.

---

Properties of meet and join:

**Closure** : For all  $x, y \in L$ ,  $\exists$  *unique*  $z$  and  $w \in L$  such that  
 $x \sqcap y = z$  and  $x \sqcup y = w$ .

**Commutative** : For all  $x, y \in L$   $x \sqcup y = y \sqcup x$  and  $x \sqcap y = y \sqcap x$ .

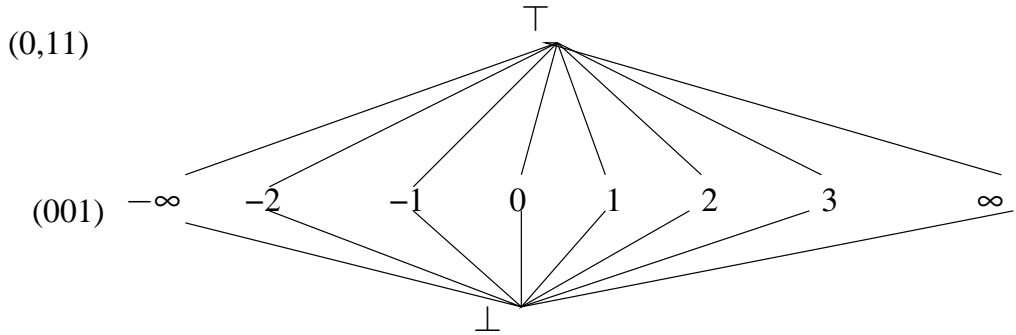
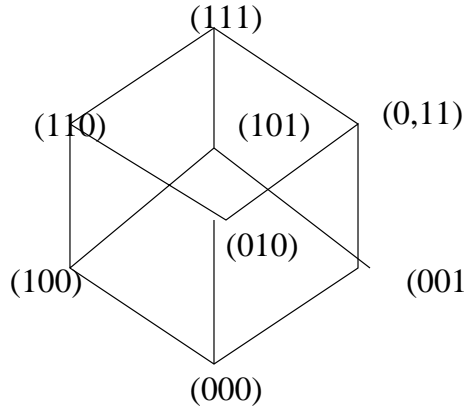
**Associative** : For all  $x, y, z \in L$   $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$  and  
 $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ .

For all  $x \in L$ ,  $x \sqcap \perp = \perp$  and  $x \sqcup \top = \top$ .

**Distributive** : (Only some lattices are.) For all  $x, y, z \in L$   
 $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$  and  $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ .

# Examples of Lattices.

---



## Lattice properties contd..

---

Meet and join induce a partial order ( $\sqsubseteq$ ) :

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

**Transitivity** : For all,  $x, y, z$  if  $x \sqsubseteq y$  and  $y \sqsubseteq z$  then  $x \sqsubseteq z$ .

**Antisymmetry** : For all,  $x, y$  if  $x \sqsubseteq y$  and  $y \sqsubseteq x$  then  $x = y$ .

**Reflexivity** : For all,  $x$  if  $x \sqsubseteq x$ .

# Constant Propagation.

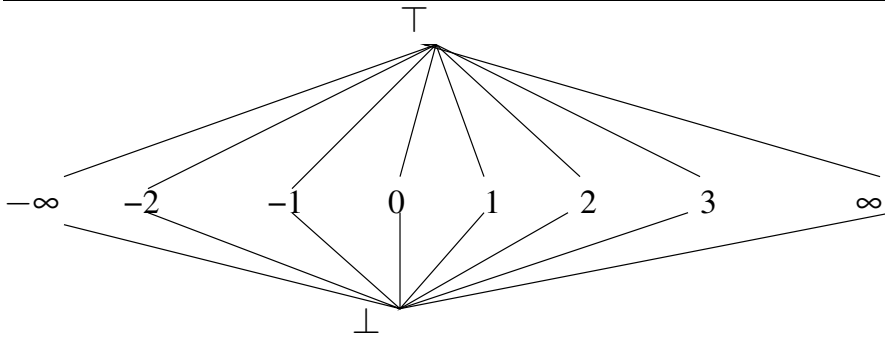
---

Goal: Discover values that are constants on *all possible executions* of a program and to propagate these constant values as far *forward* through the program as possible

**Conservative** Can discover only a subset of all the possible constants.

# Constant Propagation Lattice.

---



# Constant Propagation Lattice Meet rules.

---

$\perp$  = Constant value cannot be guaranteed.

$\top$  = May be a constant, but not yet determined.

$$x \sqcap \top = x$$

$$x \sqcap \perp = \perp$$

$$c_1 \sqcap c_1 = c_1$$

$$c_1 \sqcap c_2 = \perp$$

# Simple Constant Propagation.

---

Gary A. Kildall: A Unified Approach to Global Program Optimization - POPL 1973.

Reif, Lewis: Symbolic evaluation and the global value graph - POPL 1977.

**Simple constant** Constants that can be proved to be constant provided: No information is assumed about which direction branches will take. Only one value of each variable is maintained along each path in the program.

# Kildall's algorithm

---

- Start with an entry node in the program graph.
- Process the entry node, and produce the constant propagation information. Send it to all the immediate successors of the entry node.
- At a merge point, get an intersection of the information.
- If at any successor node, if for any variable the value is "reduced", then process the successor, similar to the processing done for entry node.