

# Dataflow Analysis

Wei Le

September 5, 2023

Acknowledgement: this lecture used slides from Ben Steenhoek, Profs Jeff Foster and Stephen Chong

# Outline

- ▶ what is dataflow analysis?
- ▶ dataflow problems
- ▶ dataflow algorithms
- ▶ dataflow analysis and bug detection

# History

1. 1970s, we want to know more about code so we can optimize the code
  - ▶ there exists any re-computation in the code?
  - ▶ there exists any useless computations?
2. Nowadays, we also want to know if the program will run correctly, what computation the legacy code is performing, ...

# What is dataflow analysis?

We want to answer questions about the data (e.g., variable, value, expressions) in a program without running the code. What are the patterns of data in a program?

- ▶ is this variable always holding a constant value?
- ▶ where you define a variable? where the variable is used
- ▶ does two variables always hold the same value?
- ▶ is this expression already "available" (computed) before reaching this program point?

# What is dataflow analysis?

1. *dataflow analysis*: determine *dataflow information* (also called variable, value and expression relationships) throughout a function or a program
2. *dataflow problem* defines which dataflow information to compute
3. dataflow analysis can be formulated into a mathematical framework
  - ▶ to generalize a set of dataflow problems so we can use one algorithm to address them all
  - ▶ to reason about the termination of dataflow analysis

# Three classical dataflow analysis problems

- ▶ *Reaching definitions* (null-pointer dereference): what *definitions* can reach a given program point
- ▶ *Available expressions* (performance issue): for each program point, what are the expressions *available*
- ▶ *Live variables* (memory leak): which variables are *alive* at a program point

See ppt slides for details

# Defining a dataflow problem: a summary

- ▶ What is the problem? (what is the dataflow information to compute at each program point?)
- ▶ The properties of dataflow problems:
  - ▶ backward or forward (determine the direction of the dataflow analysis)
  - ▶ may or must (determine how to merge values at the branches)
- ▶ Dataflow equations (local information): Gen, Kill, In, Out
- ▶ What is its applications in compiler optimization and software engineering

# Dataflow analysis



# Generalizing dataflow analysis

- ▶ **Goal:** solving dataflow equations and determining dataflow information at all the program points in the program
  - ▶ local information can be propagated along program control flow to influence other nodes
  - ▶ will the dataflow information be stabilized, and the analysis can be terminated?
- ▶ **Framework:** a set of dataflow propagation algorithms for a set of dataflow problems
- ▶ **Key of the algorithm:**
  - ▶ dataflow equation computes dataflow information locally
  - ▶ dataflow algorithms: connect dataflow information globally, especially stabilizing the dataflow solutions in presence of loops via, e.g., *fixpoint* (iterating through the loop until dataflow information no longer changes)


# Forward must data flow algorithm

```
Out(s) =  $\top$  for all statements s
W := { all statements }           (worklist)
repeat {
    Take s from W
    In(s) :=  $\bigcap_{s' \in \text{pred}(s)} \text{Out}(s')$ 
    temp := Gen(s)  $\cup$  (In(s) - Kill(s))
    if (temp  $\neq$  Out(s)) {
        Out(s) := temp
        W := W  $\cup$  succ(s)
    }
} until W =  $\emptyset$ 
```

# Forward data flow again

```
Out(s) =  $\top$  for all statements s
W := { all statements }
repeat {
  Take s from W
  temp :=  $f_s(\prod_{s' \in \text{pred}(s)} \text{Out}(s'))$ 
  if (temp  $\neq$  Out(s)) {
    Out(s) := temp
    W := W  $\cup$  succ(s)
  }
} until W =  $\emptyset$ 
```

*Transfer function for statement s*



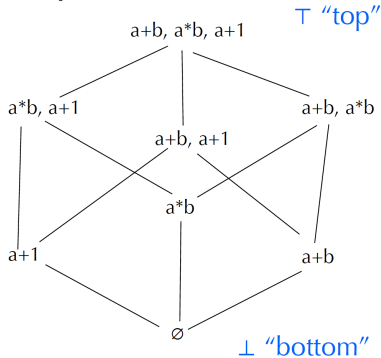
# Dataflow analysis and lattice [Kildall:1973] [Kam:1976]

For analyzing termination and for generalizing the algorithms to a set of dataflow problems

- ▶ *Lattice ( $L$ )*: the set of elements plus the order of these elements, it has a upper bound and a lower bound
- ▶ *Set*: each dataflow problem has a lattice that describes the domain of the dataflow solutions. That is, at each program point, the dataflow fact is an element of a lattice
- ▶ *Order*: "order" among dataflow facts : dataflow set  $A$  is a conservative approximation of dataflow set  $B$ , noted as  $A < B$ , bottom – the most conservative solution
- ▶ *transfer (flow) function* for dataflow problem: how each edge in lattice affects the dataflow set  $F : L \rightarrow L$
- ▶ *merge function* for dataflow problem: reduce to meet operator  $\sqcap$  ( $\wedge$ ) on lattice

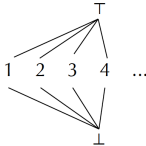
# Data flow facts and lattices

- Typically, data flow facts form lattices
- E.g., available expressions



# Useful lattices

- $(2^S, \subseteq)$  forms a lattice for any set  $S$ 
  - $2^S$  is powerset of  $S$ , the set of all subsets of  $S$ .
- If  $(S, \leq)$  is a lattice, so is  $(S, \geq)$ 
  - i.e., can “flip” the lattice
- Lattice for constant propagation



**Example:** Liveness analysis with 3 variables

$$S = \{v1, v2, v3\}$$

– V:  $2^S = \{\{v1, v2, v3\}, \{v1, v2\}, \{v1, v3\}, \{v2, v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset\}$

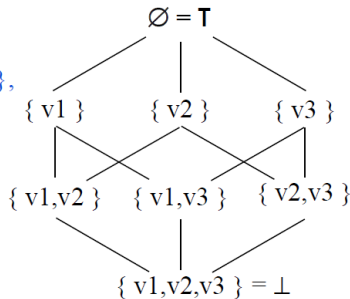
– Meet ( $\wedge$ ):  $\cup$

–  $\leq$ :  $\supseteq$

– Top(T):  $\emptyset$

– Bottom ( $\perp$ ):  $\mathbf{v}$

– F:  $\{f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n\}$



# Termination

- We know the algorithm terminates
- In each iteration, either  $W$  gets smaller, or  $\text{Out}(s)$  decreases for some  $s$ 
  - Since function is monotonic
- Lattice has only finite height, so for each  $s$ ,  $\text{Out}(s)$  can decrease only finitely often

```
Out(s) =  $\top$  for all statements  $s$ 
W := { all statements }
repeat {
  Take  $s$  from  $W$ 

  In(s) :=  $\bigcap_{s' \in \text{pred}(s)} \text{Out}(s')$ 
  temp := Gen(s)  $\cup$  (In(s) - Kill(s))
  if (temp  $\neq$  Out(s)) {
    Out(s) := temp
    W := W  $\cup$  succ(s)
  }
} until W =  $\emptyset$ 
```



# Termination

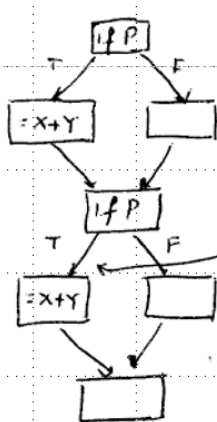
- A **descending chain** in a lattice is a sequence  $x_0 < x_1 < \dots$
- The **height of a lattice** is the length of the longest descending chain in the lattice
- Then, dataflow must terminate in  $O(nk)$  time
  - $n$  = # of statements in program
  - $k$  = height of lattice
  - assumes meet operation and transfer function takes  $O(1)$  time

# Dataflow analysis precision - conservative analysis

- ▶ For compiler optimizations, the dataflow facts we compute should definitely be true (not simply possibly true).
- ▶ However, we may miss optimization opportunities (not able to compute a complete and correct solution): two main reasons
  - ▶ Control Flow
  - ▶ Pointers & Aliasing

# Dataflow analysis precision-control flow

We assume that all paths are executable; however, some may be infeasible.



$X+Y$  is always available if we exclude infeasible paths.

# Dataflow analysis precision-pointer analysis

we may not know what a pointer points to:

1.  $X = 5$
2.  $*p = \dots$  //  $p$  may or may not point to  $X$
3.  $\dots = X$

**Constant propagation:** assume  $p$  does point to  $X$  (i.e., in statement 3,  $X$  cannot be replaced by 5).

**Dead Code Elimination:** assume  $p$  does not point to  $X$  (i.e., statement 1 cannot be deleted).

# Dataflow analysis implementation

- ▶ Does a definition reach a point ? T or F, each variable definition is a bit, each program point has a bitvector
- ▶ Is an expression available ? T or F, each expression is a bit, each program point has a bitvector
- ▶ Is a variable live ? T or F, each variable is a bit, each program point has a bitvector

Intersection and union operations in dataflow equations can be implemented using bitwise *and* & *or* operations.

# Practical implementation

- Data flow facts are assertions that are true or false at a program point
- Can represent set of facts as bit vector
  - Fact  $i$  represented by bit  $i$
  - Intersection=bitwise and, union=bitwise or, etc
- “Only” a constant factor speedup
  - But very useful in practice

# Dataflow analysis for bug detection

problem reduction: what are the dataflow facts?

- ▶ How to detect uninitialized variables?
- ▶ How to detect memory leaks?

make the analysis more precise and less false positives

- ▶ How to detect infeasible paths
- ▶ How to make it more precise by considering pointer aliasing information
- ▶ How to track inter-procedural bugs?
- ▶ ...

# Static analysis for bug detection: steps

- ▶ construct a cfg
- ▶ map to a dataflow problem
- ▶ dataflow analysis
- ▶ extend interprocedurally
- ▶ further improve precision: add pointer analysis and infeasible paths detection



# Further Reading

Lattice Theory by Patrick Cousot

Data flow analysis in Principles of Program Analysis