Dataflow Analysis

Wei Le

September 7, 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 *dataflow facts* (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

Dataflow analysis

Generalizing dataflow analysis

- **Goal**: solving dataflow equations and determining dataflow information at all the program points in the program
- **Framework**: an algorithm for a set of dataflow problems of the same category
- **Key of the algorithm**:
  - datafow equation computes dataflow information locally
  - local information can be propagated along program control flow to influence other nodes
  - dataflow algorithms: connect dataflow information globally
  - will the dataflow analysis be terminated, especially in presense of loops?
  - the algorith terminates when dataflow information is stabilized via *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
    $In(s) := \bigcap_{s' \in pred(s)} Out(s')$
    $temp := Gen(s) \cup (In(s) - Kill(s))$
    if (temp != Out(s)) {
        Out(s) := temp
        $W := W \cup succ(s)$
    }
} until $W = \varnothing$

Forward data flow again

$Out(s) = \top$ for all statements s
W := { all statements }
repeat {
    Take s from W
    temp := $\boxed{f_s(\sqcap_{s' \in pred(s)} Out(s'))}$
    if (temp != Out(s)) {
        Out(s) := temp
        W := W $\cup$ succ(s)
    }
} until W = $\varnothing$

Transfer function for statement s

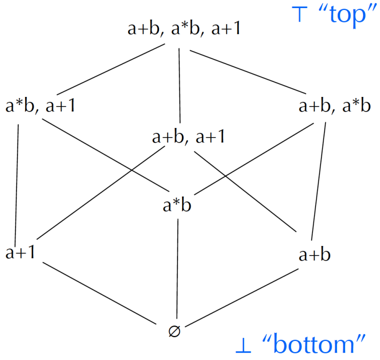
Termination: reduce dataflow analysis to lattice computation [Kildall:1973] [Kam:1976]

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

- *Lattice (L)*: is a set; the elements in this set has an order; the set has a upper bound and a lower bound
- Set: each dataflow problem has a lattice. At each program point, the dataflow information/fact is an element of a lattice
- Order: "order" among dataflow facts : dataflow fact A is a conservative approximation of dataflow fact B, noted as $A < B$, bottom – the most conservative solution
- Edge: how *transfer function* affects the dataflow fact
- *merge function*: reduce to meet operator $\sqcap$ ($\wedge$) on lattice

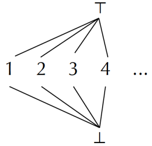
Data flow facts and lattices

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



Useful lattices

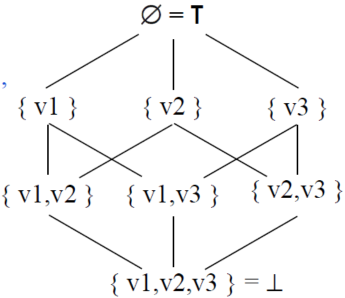
- $(2^S, \subseteq)$ forms a lattice for any set S
  - 2^S is powerset of S, the set of all subsets of S.
- If $(S, \leq)$ is a lattice, so is $(S, \geq)$
  - i.e., can "flip" the lattice
- Lattice for constant propagation



Example: Liveness analysis with 3 variables

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

- V: $2^S = \{\{v1,v2,v3\}, \{v1,v2\},\{v1,v3\},\{v2,v3\}, \{v1\},\{v2\},\{v3\}, \varnothing\}$
- Meet (^): $\cup$
  - $\leq$: $\supseteq$
  - Top(T): $\varnothing$
  - Bottom ($\perp$): υ
- F: $\{f_n(X) = Gen_n \cup (X - Kill_n), \forall n\}$



Termination

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

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

    In(s) := ∩_{s' ∈ pred(s)} Out(s')
    temp := Gen(s) ∪ (In(s) - Kill(s))
    if (temp != Out(s)) {
        Out(s) := temp
        W := W ∪ succ(s)
    }
} until W = ∅
```

Termination

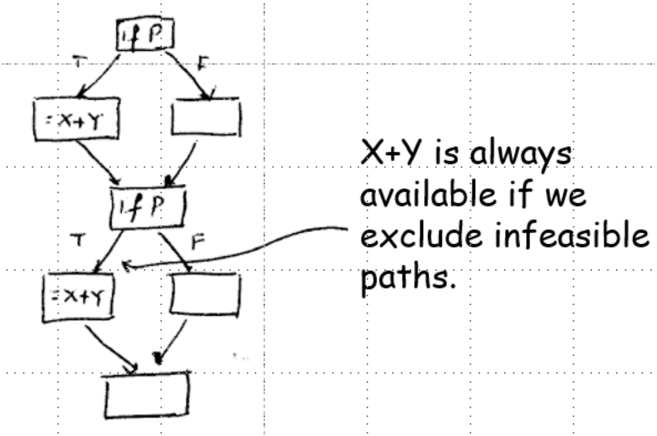
- A **descending chain** in a lattice is a sequence $x_0 < x_1 < \ldots$
- The **height of a lattice** is the length of the longest descending chain in the lattice
- Then, dataflow must terminate in O(nk) time
  - n = # of statements in program
  - k = height of lattice
  - assumes meet operation and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



Dataflow analysis precision-pointer analysis

we may not know what a pointer points to:

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

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