Dataflow Analysis

Wei Le

February 8, 2019

What is dataflow analysis?

1. History (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. *dataflow analysis*: global analysis of variables and expressions relationships
   - how the variables are defined and used?
   - is this expression already "available"? at this program point?
3. formulated into a mathematical framework, e.g., using *lattice*
   - to reason about the termination of dataflow analysis
   - to generalize a set of dataflow problems so we can use one algorithm to address them all
4. is a type of *abstraction*
   - using abstract values to represent answers (related to *abstract interpretation*)
   - merge paths, fix points for loops (related to *model checking*)

Three classical dataflow analysis problems

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

See ppt slides for example details

Generalizing dataflow analysis

An instance of dataflow problem:

- What is the problem? (what are the properties/dataflow facts 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

Generalizing Dataflow Analysis

- **Goal**: solving dataflow equations, determine dataflow facts/program point for the whole program
- **Framework**: a set of data propagation algorithms for a set of dataflow problems
- **Key of the algorithm**:
  - datafow equation computes dataflow facts locally
  - dataflow algorithms: Connect dataflow facts globally, especially stabilizing the dataflow facts/solutions in presence of loops via, e.g., fixpoint
- **Data structure for efficiency**: bit vector to represent the binary information

Representing Dataflow Facts

- Does a definition reach a point ? T or F, each definition is a bit, each program point has a bitvector
- Is an expression available/very busy ? 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 can be implemented using bitwise *and* & *or* operations.

Dataflow Algorithms – Solving Dataflow Equations

Solving dataflow equations for all program points: performing dataflow analysis for the entire program, propagating dataflow until fix points are reached (stabilize via loops)

Iterative Approach

- initialize sets
- iterate over the sets till they stabilize

Example: Forward problem (Available Expressions)

```
IN[B0] = ∅ ;   OUT[B0] = GEN[B0]
For i = 1 to N do  OUT[Bi] = {all expressions} - KILL[Bi]
change = true
while change do
    change = false
    for each block B ≠ B0 do
        OLDOUT = OUT[B]
        IN[B] = ∩ OUT[P]
              P ∈ pred(B)
        OUT[B] = GEN[B] ∪ (IN[B] - KILL[B])
        IF OUT[B] ≠ OLDOUT then change = true
    endfor
endwhile
```

$\cap$ - start with largest estimate & iteratively shrink the solution till it stabilizes.

Iterative Approach

Example- backward problem (live variables)

```
for i = 1 to N do  IN[B_i] = GEN[B_i]
OUT[B_exit] = ∅
change = true
while change do
    change = false
    for each block B do
        OLDIN = IN[B]
        OUT[B] =   ∪      IN[S]
                S∈SUCC(B)
        IN[B] = GEN[B] ∪ (OUT[B] - KILL[B])
        If OLDIN ≠ IN[B] then change = true
    end for
endwhile
```

Alternative Approach- Worklist Algorithm

Example - backward problem (very busy expressions)

for $i = 1$ to N do $IN[B_i] = \{\text{All expressions}\} - KILL[B_i]$

$OUT[B_{exit}] = \emptyset$

Worklist ← All blocks

while Worklist $\neq \emptyset$ do

  get B from worklist

  OLDIN = IN[B]

  $OUT[B] = \bigcap_{S \in succ(B)} IN[S]$

  $IN[B] = GEN(B) \cup (OUT[B] - KILL[B])$

  If OLDIN $\neq$ IN[B] then

    Add Pred (B) to worklist

endwhile

$\bigcap$ - start with largest solution & keep iterating till it stops shrinking.

Conservative Analysis

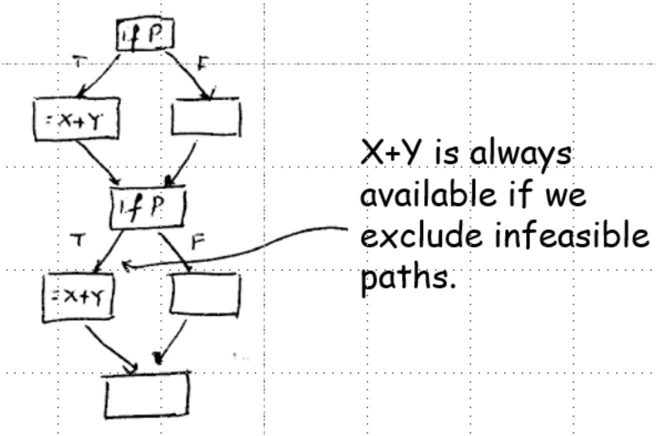
For compilation optimizations, the data flow facts we compute should definitely be true (not simply possibly true).

Two main reasons that cause results of analysis to be conservative:

- Control Flow
- Pointers & Aliasing

Conservative Analysis - Control Flow

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



Conservative Analysis - 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).

Formalize Dataflow Problems

- *Lattice (L)*: the set of elements plus the order of these elements, it has a upper bound and a lower bound
- Operators on lattice:
  - join operator – the least upper bound $\sqcup(\{1\},\{2\},\{3\}) = \{1,2,3\}$
  - meet operator – the greatest lower bound, $\sqcap(\{1\},\{1,2\}) = \{1\}$

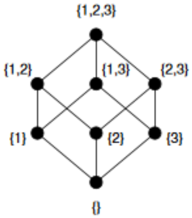


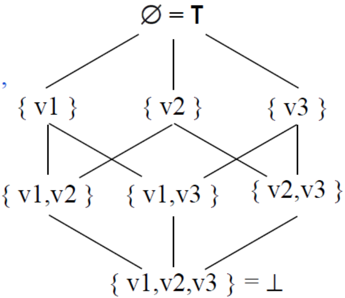
Figure 1: A Hasse diagram.

Formalize Dataflow Problems [Kildall:1973] [Kam:1976]

- each dataflow problem has a lattice that describes the domain of the dataflow facts. That is, at each program point, the dataflow fact is an element of a lattice
- "order" in dataflow: element a is a conservative approximation of b, a $<$ b
- *flow function for dataflow problem*: how each node affects the dataflow fact $F : L \to L$
- *merge functions for dataflow problem*: reduce to meet or join operators on lattice

Example: Liveness analysis with 3 variables
$S = \{v1, v2, v3\}$

- V: $\mathbf{2^S} = \{\{v1,v2,v3\}, \{v1,v2\},\{v1,v3\},\{v2,v3\}, \{v1\},\{v2\},\{v3\}, \emptyset\}$
- Meet (^): $\cup$
  - $\leq$: $\supseteq$
  - Top(T): $\emptyset$
  - Bottom (⊥): $\boldsymbol{\upsilon}$
- F: $\{f_n(X) = Gen_n \cup (X - Kill_n), \forall n\}$



Further Reading

Lattice Theory by Patrick Cousot
Data flow analysis in Principles of Program Analysis