Dataflow Analysis

Wei Le

February 4, 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)

$IN[B_0] = \emptyset$; $OUT[B_1] = GEN[B_1]$

For $i = 1$ to N do $OUT[B_i] = \{all\ expressions\} - KILL[B_i]$

change = true

while change do

    change = false

    for each block $B \neq B_0$ do

        OLDOUT = $OUT[B]$

        $IN[B] = \bigcap_{P \in pred(B)} OUT[P]$

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

        IF $OUT[B] \neq OLDOUT$ then change = true

    end for

  end while

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

Iterative Approach

Example - backward problem (live variables)

for $i = 1$ to N do $IN[B_i] = GEN[B_i]$

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

change = true

while change do

    change = false

    for each block B do

        OLDIN = $IN[B]$

        $OUT[B] = \bigcup_{s \in succ(B)} IN[s]$

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

        If OLDIN $\neq IN[B]$ then change = true

    end for

end while

Alternative Approach: Worklist Algorithm

Example - backward problem (very busy expressions)

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

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

Worklist $\leftarrow$ All blocks

while Worklist $\neq \emptyset$ do

    get B from Worklist

    $OLDIN = IN[B]$

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

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

    If $OLDIN \neq IN[B]$ then

        Add $Pred(B)$ to Worklist

endwhile

$\cap$ - start with largest solution & keep iterating till it stops shrinking.
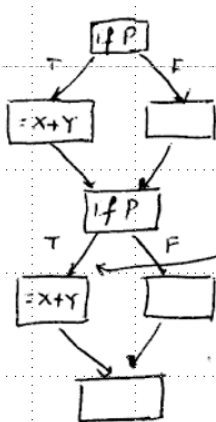
Conservative Analysis

Optimizations that we apply must be Safe =¿ 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 (but not exact):

► Control Flow
► Pointers & Aliasing

Conservative Analysis - Control Flow

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

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

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
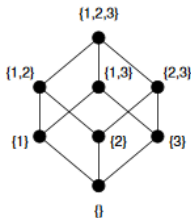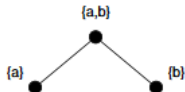

Figure 1: A Hasse diagram.

(upperbound, lowerbound)

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

- each dataflow problem has a lattice that describes the domain of the dataflow facts
- Lattice and dataflow analysis: at each program point, the dataflow fact is an element of a lattice
- "order" in dataflow, e.g., element a is a conservative approximation of b, a $<$ b
- *flow function for dataflow problem*: how each node affects the dataflow fact $F : L \rightarrow L$
- *merge functions for dataflow problem*: *meet operator* in lattice operations: e.g., join operator – the least upper bound $\sqcup(\{1\}, \{2\}, \{3\}) = \{1, 2, 3\}$; meet operator – the greatest lower bound, $\sqcap(\{1\}, \{1, 2\}) = \{1\}$

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 (^): $\cup$

  – ≤: $\supseteq$
  – Top(⊤): $\varnothing$
  – Bottom (⊥): υ

– F: {$f_n(X) = Gen_n \cup (X - Kill_n)$, $\forall n$}

$\varnothing$ = ⊤

{ v1 }   { v2 }   { v3 }

{ v1,v2 }   { v1,v3 }   { v2,v3 }

{ v1,v2,v3 } = ⊥
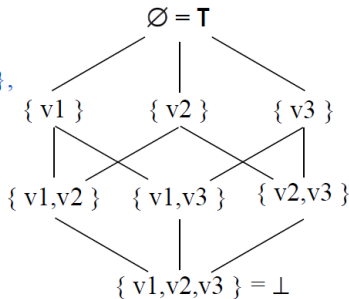
Further Reading

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