

6.035

Fall 2006

Foundations of Dataflow Analysis

Dataflow Analysis

- Compile-Time Reasoning About
- Run-Time Values of Variables or Expressions
- At Different Program Points
 - Which assignment statements produced value of variable at this point?
 - Which variables contain values that are no longer used after this program point?
 - What is the range of possible values of variable at this program point?

Samir Amarasinghe

2

6.035 ©MIT Fall 1998

Program Representation

- Control Flow Graph
 - Nodes N – statements of program
 - Edges E – flow of control
 - $\text{pred}(n)$ = set of all predecessors of n
 - $\text{succ}(n)$ = set of all successors of n
 - Start node n_0
 - Set of final nodes N_{final}

Samir Amarasinghe

3

6.035 ©MIT Fall 1998

Program Points

- One program point before each node
- One program point after each node
- Join point – point with multiple predecessors
- Split point – point with multiple successors

Samir Amarasinghe

4

6.035 ©MIT Fall 1998

Basic Idea

- Information about program represented using values from algebraic structure called lattice
- Analysis produces lattice value for each program point
- Two flavors of analysis
 - Forward dataflow analysis
 - Backward dataflow analysis

Samir Amarasinghe

5

6.035 ©MIT Fall 1998

Partial Orders

- Set P
- Partial order \leq such that $\forall x, y, z \in P$
 - $x \leq x$ (reflexive)
 - $x \leq y$ and $y \leq x$ implies $x = y$ (asymmetric)
 - $x \leq y$ and $y \leq z$ implies $x \leq z$ (transitive)
- Can use partial order to define
 - Upper and lower bounds
 - Least upper bound
 - Greatest lower bound

Samir Amarasinghe

6

6.035 ©MIT Fall 1998

Upper Bounds

- If $S \subseteq P$ then
 - $x \in P$ is an upper bound of S if $\forall y \in S. y \leq x$
 - $x \in P$ is the least upper bound of S if
 - x is an upper bound of S , and
 - $x \leq y$ for all upper bounds y of S
 - \vee - join, least upper bound, lub, supremum, sup
 - $\vee S$ is the least upper bound of S
 - $x \vee y$ is the least upper bound of $\{x, y\}$

Samir Amarasinghe

7

6.035 ©MIT Fall 1998

Lower Bounds

- If $S \subseteq P$ then
 - $x \in P$ is a lower bound of S if $\forall y \in S. x \leq y$
 - $x \in P$ is the greatest lower bound of S if
 - x is a lower bound of S , and
 - $y \leq x$ for all lower bounds y of S
 - \wedge - meet, greatest lower bound, glb, infimum, inf
 - $\wedge S$ is the greatest lower bound of S
 - $x \wedge y$ is the greatest lower bound of $\{x, y\}$

Samir Amarasinghe

8

6.035 ©MIT Fall 1998

Covering

- $x < y$ if $x \leq y$ and $x \neq y$
- x is covered by y (y covers x) if
 - $x < y$, and
 - $x \leq z < y$ implies $x = z$
- Conceptually, y covers x if there are no elements between x and y

Samir Amarasinghe

9

6.035 ©MIT Fall 1998

Example

- $P = \{000, 001, 010, 011, 100, 101, 110, 111\}$
(standard boolean lattice, also called hypercube)
- $x \leq y$ if $(x \text{ bitwise and } y) = x$



Hasse Diagram

- If y covers x
 - Line from y to x
 - y above x in diagram

Samir Amarasinghe

10

6.035 ©MIT Fall 1998

Lattices

- If $x \wedge y$ and $x \vee y$ exist for all $x, y \in P$, then P is a lattice.
- If $\wedge S$ and $\vee S$ exist for all $S \subseteq P$, then P is a complete lattice.
- All finite lattices are complete
- Example of a lattice that is not complete
 - Integers \mathbb{I}
 - For any $x, y \in \mathbb{I}$, $x \vee y = \max(x, y)$, $x \wedge y = \min(x, y)$
 - But $\vee \mathbb{I}$ and $\wedge \mathbb{I}$ do not exist
 - $\mathbb{I} \cup \{+\infty, -\infty\}$ is a complete lattice

Samir Amarasinghe

11

6.035 ©MIT Fall 1998

Top and Bottom

- Greatest element of P (if it exists) is top
- Least element of P (if it exists) is bottom (\perp)

Samir Amarasinghe

12

6.035 ©MIT Fall 1998

Connection Between \leq , \wedge , and \vee

- The following 3 properties are equivalent:
 - $x \leq y$
 - $x \vee y = y$
 - $x \wedge y = x$

Lattices as Algebraic Structures

- Have defined \vee and \wedge in terms of \leq
- Will now define \leq in terms of \vee and \wedge
 - Start with \vee and \wedge as arbitrary algebraic operations that satisfy associative, commutative, idempotence, and absorption laws
 - Will define \leq using \vee and \wedge
 - Will show that \leq is a partial order
- Intuitive concept of \vee and \wedge as information combination operators (or, and)

Algebraic Properties of Lattices

Assume arbitrary operations \vee and \wedge such that

- $(x \vee y) \vee z = x \vee (y \vee z)$ (associativity of \vee)
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (associativity of \wedge)
- $x \vee y = y \vee x$ (commutativity of \vee)
- $x \wedge y = y \wedge x$ (commutativity of \wedge)
- $x \vee x = x$ (idempotence of \vee)
- $x \wedge x = x$ (idempotence of \wedge)
- $x \vee (x \wedge y) = x$ (absorption of \vee over \wedge)
- $x \wedge (x \vee y) = x$ (absorption of \wedge over \vee)

Connection Between \wedge and \vee

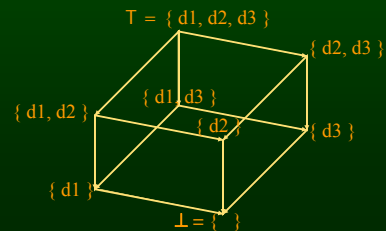
- $x \vee y = y$ if and only if $x \wedge y = x$
- Proof of $x \vee y = y$ implies $x = x \wedge y$
 - $x = x \wedge (x \vee y)$ (by absorption)
 - $= x \wedge y$ (by assumption)
- Proof of $x \wedge y = x$ implies $y = x \vee y$
 - $y = y \vee (y \wedge x)$ (by absorption)
 - $= y \vee (x \wedge y)$ (by commutativity)
 - $= y \vee x$ (by assumption)
 - $= x \vee y$ (by commutativity)

Chains

- A set S is a chain if $\forall x, y \in S. y \leq x$ or $x \leq y$
- P has no infinite chains if every chain in P is finite

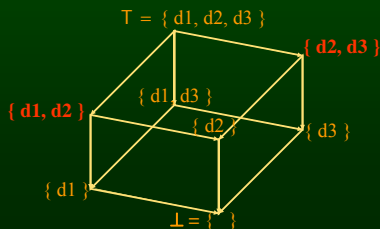
Lattice

- Example: the lattice for the reaching definition problem when there are only 3 definitions



Meet and Join Operations

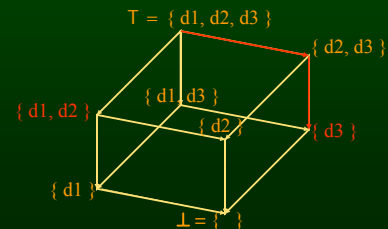
$$\{d1, d2\} \wedge \{d2, d3\} = ???$$



Samir Amaralgahe 19 6.035 ©MIT Fall 1998

Meet and Join Operations

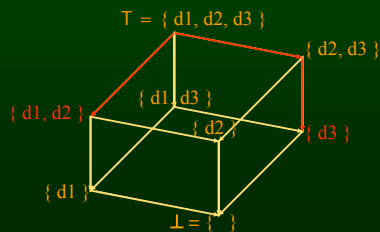
$$\{d1, d2\} \vee \{d3\} = ???$$



Samir Amaralgahe 20 6.035 ©MIT Fall 1998

Meet and Join Operations

$$\{d1, d2\} \vee \{d3\} = ???$$



Samir Amaralgahe 21 6.035 ©MIT Fall 1998

Application to Dataflow Analysis

- Dataflow information will be lattice values
 - Transfer functions operate on lattice values
 - Solution algorithm will generate increasing sequence of values at each program point
 - Ascending chain condition will ensure termination
- Will use \vee to combine values at control-flow join points

Samir Amaralgahe 22 6.035 ©MIT Fall 1998

Transfer Functions

- Transfer function $f: P \rightarrow P$ for each node in control flow graph
- f models effect of the node on the program information

Samir Amaralgahe 23 6.035 ©MIT Fall 1998

Transfer Functions

Each dataflow analysis problem has a set F of transfer functions $f: P \rightarrow P$

- Identity function $I \in F$
- F must be closed under composition:
 - $\forall f, g \in F$, the function $h = \lambda x. f(g(x)) \in F$
- Each $f \in F$ must be monotone:
 - $x \leq y$ implies $f(x) \leq f(y)$
- Sometimes all $f \in F$ are distributive:
 - $f(x \vee y) = f(x) \vee f(y)$
- Distributivity implies monotonicity

Samir Amaralgahe 24 6.035 ©MIT Fall 1998

Putting Pieces Together

- Forward Dataflow Analysis Framework
- Simulates execution of program forward with flow of control

Forward Dataflow Analysis

- Simulates execution of program forward with flow of control
- For each node n , have
 - in_n – value at program point before n
 - out_n – value at program point after n
 - f_n – transfer function for n (given in_n , computes out_n)
- Require that solution satisfy
 - $\forall n. out_n = f_n(in_n)$
 - $\forall n \neq n_0. in_n = \vee \{ out_m . m \text{ in } pred(n) \}$
 - $in_{n_0} = I$
 - Where I summarizes information at start of program

Dataflow Equations

- Compiler processes program to obtain a set of dataflow equations
 - $out_n := f_n(in_n)$
 - $in_n := \vee \{ out_m . m \text{ in } pred(n) \}$
- Conceptually separates analysis problem from program

Worklist Algorithm for Solving Forward Dataflow Equations

```
for each  $n$  do
   $out_n := f_n(I)$ 
 $in_{n_0} := I$ ;
 $out_{n_0} := f_{n_0}(I)$ 
worklist :=  $N - \{ n_0 \}$ 
while worklist  $\neq \emptyset$  do
  remove a node  $n$  from worklist
   $in_n := \vee \{ out_m . m \text{ in } pred(n) \}$ 
   $out_n := f_n(in_n)$ 
  if  $out_n$  changed then
    worklist := worklist  $\cup$  succ( $n$ )
```

Correctness Argument

- Why result satisfies dataflow equations
- Whenever process a node n , set $out_n := f_n(in_n)$
Algorithm ensures that $out_n = f_n(in_n)$
- Whenever out_m changes, put $succ(m)$ on worklist. Consider any node $n \in succ(m)$. It will eventually come off worklist and algorithm will set
 - $in_n := \vee \{ out_m . m \text{ in } pred(n) \}$
 - to ensure that $in_n = \vee \{ out_m . m \text{ in } pred(n) \}$
- So final solution will satisfy dataflow equations

Termination Argument

- Why does algorithm terminate?
- Sequence of values taken on by in_n or out_n is a chain. If values stop increasing, worklist empties and algorithm terminates.
- If lattice has ascending chain property, algorithm terminates
 - Algorithm terminates for finite lattices
 - For lattices without ascending chain property, use widening operator

Widening Operators

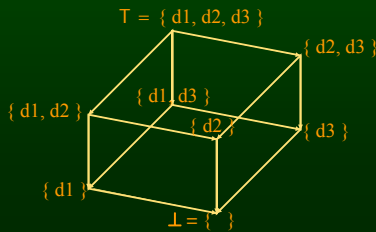
- Detect lattice values that may be part of infinitely ascending chain
- Artificially raise value to least upper bound of chain
- Example:
 - Lattice is set of all subsets of integers
 - Could be used to collect possible values taken on by variable during execution of program
 - Widening operator might raise all sets of size n or greater to TOP (likely to be useful for loops)

Reaching Definitions

- P = powerset of set of all definitions in program (all subsets of set of definitions in program)
- $\vee = \cup$ (order is \subseteq)
- $\perp = \emptyset$
- $I = \text{in}_{n0} = \perp$
- F = all functions f of the form $f(x) = a \cup (x-b)$
 - b is set of definitions that node kills
 - a is set of definitions that node generates
- General pattern for many transfer functions
 - $f(x) = \text{GEN} \cup (x\text{-KILL})$

Lattice

- Example: the lattice for the reaching definition problem when there are only 3 definitions



Does Reaching Definitions Framework Satisfy Properties?

- \subseteq satisfies conditions for \leq
 - $x \subseteq y$ and $y \subseteq z$ implies $x \subseteq z$ (transitivity)
 - $x \subseteq y$ and $y \subseteq x$ implies $y = x$ (asymmetry)
 - $x \subseteq x$ (idempotence)
- F satisfies transfer function conditions
 - $\lambda x. \emptyset \cup (x - \emptyset) = \lambda x. x \in F$ (identity)
 - Will show $f(x \cup y) = f(x) \cup f(y)$ (distributivity)

$$f(x) \cup f(y) = (a \cup (x - b)) \cup (a \cup (y - b))$$

$$= a \cup (x - b) \cup (y - b) = a \cup ((x \cup y) - b)$$

$$= f(x \cup y)$$

Does Reaching Definitions Framework Satisfy Properties?

- What about composition?
 - Given $f_1(x) = a_1 \cup (x - b_1)$ and $f_2(x) = a_2 \cup (x - b_2)$
 - Must show $f_1(f_2(x))$ can be expressed as $a \cup (x - b)$

$$f_1(f_2(x)) = a_1 \cup ((a_2 \cup (x - b_2)) - b_1)$$

$$= a_1 \cup ((a_2 - b_1) \cup ((x - b_2) - b_1))$$

$$= (a_1 \cup (a_2 - b_1)) \cup ((x - b_2) - b_1)$$

$$= (a_1 \cup (a_2 - b_1)) \cup (x - (b_2 \cup b_1))$$
 - Let $a = (a_1 \cup (a_2 - b_1))$ and $b = b_2 \cup b_1$
 - Then $f_1(f_2(x)) = a \cup (x - b)$

General Result

- All GEN/KILL transfer function frameworks satisfy
- Identity
 - Distributivity
 - Composition
- Properties

Available Expressions

- P = powerset of set of all expressions in program (all subsets of set of expressions)
- $\vee = \cap$ (order is \supseteq)
- $\perp = P$
- $I = in_{n_0} = \emptyset$
- F = all functions f of the form $f(x) = a \cup (x-b)$
 - b is set of expressions that node kills
 - a is set of expressions that node generates
- Another GEN/KILL analysis

Samir Amarasinghe

37

6.035 ©MIT Fall 1998

Concept of Conservatism

- Reaching definitions use \cup as join
 - Optimizations must take into account all definitions that reach along ANY path
- Available expressions use \cap as join
 - Optimization requires expression to reach along ALL paths
- Optimizations must conservatively take all possible executions into account. Structure of analysis varies according to way analysis used.

Samir Amarasinghe

38

6.035 ©MIT Fall 1998

Backward Dataflow Analysis

- Simulates execution of program backward against the flow of control
- For each node n , have
 - in_n – value at program point before n
 - out_n – value at program point after n
 - f_n – transfer function for n (given out_n , computes in_n)
- Require that solution satisfies
 - $\forall n. in_n = f_n(out_n)$
 - $\forall n \notin N_{final}. out_n = \vee \{ in_m . m \text{ in } succ(n) \}$
 - $\forall n \in N_{final} = out_n = O$
 - Where O summarizes information at end of program

Samir Amarasinghe

39

6.035 ©MIT Fall 1998

Worklist Algorithm for Solving Backward Dataflow Equations

```

for each n do
   $in_n := f_n(\perp)$ 
for each  $n \in N_{final}$  do
   $out_n := O$ 
   $in_n := f_n(O)$ 
worklist :=  $N - N_{final}$ 
while worklist  $\neq \emptyset$  do
  remove a node  $n$  from worklist
   $out_n := \vee \{ in_m . m \text{ in } succ(n) \}$ 
   $in_n := f_n(out_n)$ 
  if  $in_n$  changed then
    worklist := worklist  $\cup$  pred(n)
    
```

Samir Amarasinghe

40

6.035 ©MIT Fall 1998

Live Variables

- P = powerset of set of all variables in program (all subsets of set of variables in program)
- $\vee = \cup$ (order is \subseteq)
- $\perp = \emptyset$
- $O = \emptyset$
- F = all functions f of the form $f(x) = a \cup (x-b)$
 - b is set of variables that node kills
 - a is set of variables that node reads

Samir Amarasinghe

41

6.035 ©MIT Fall 1998

Pessimistic vs. Optimistic Analyses

- Available expressions is optimistic (for common sub-expression elimination)
 - Assumes expressions are available at start of analysis
 - Analysis eliminates all that are not available
 - If analysis result $in_n \leq e$, can use e for CSE
 - Cannot stop analysis early and use current result
- Live variables is pessimistic (for dead code elimination)
 - Assumes all variables are live at start of analysis
 - Analysis finds variables that are dead
 - If $e \leq$ analysis result in_n , can use e for dead code elimination
 - Can stop analysis early and use current result
- Formal dataflow setup same for both analyses
- Optimism/pessimism depends on intended use

Samir Amarasinghe

42

6.035 ©MIT Fall 1998

Static Single Assignment (SSA) Form

- Each definition has a unique variable name
 - Original name + a version number
- Each use refers to a definition by name
- What about multiple possible definitions?
 - Add special merge nodes so that there can be only a single definition (Φ functions)

Suman Aravamudan

43

6.035 ©MIT Fall 1998

Static Single Assignment (SSA) Form

```
a = 1
b = a + 2
c = a + b
a = a + 1
d = a + b
```



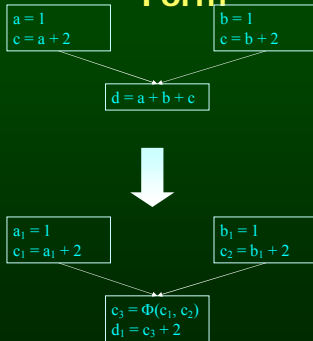
```
a1 = 1
b1 = a1 + 2
c1 = a1 + b1
a2 = a1 + 1
d1 = a2 + b1
```

Suman Aravamudan

44

6.035 ©MIT Fall 1998

Static Single Assignment (SSA) Form



Suman Aravamudan

45

6.035 ©MIT Fall 1998

Summary

- Formal dataflow analysis framework
 - Lattices, partial orders
 - Transfer functions, joins and splits
 - Dataflow equations and fixed point solutions

Suman Aravamudan

46

6.035 ©MIT Fall 1998