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?
Program Representation

• Control Flow Graph
  – Nodes N – statements of program
  – Edges E – flow of control
    • pred(n) = set of all predecessors of n
    • succ(n) = set of all successors of n
  – Start node $n_0$
  – Set of final nodes $N_{final}$
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
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
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
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 $S$
- $\lor$ - join, least upper bound, lub, supremum, sup
  - $\lor$ $S$ is the least upper bound of $S$
  - $x \lor y$ is the least upper bound of $\{x,y\}$
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 $S$
  - $\land$ - meet, greatest lower bound, glb, infimum, inf
    • $\land S$ is the greatest lower bound of $S$
    • $x \land y$ is the greatest lower bound of $\{x,y\}$
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 $y$
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
Lattices

• If $x \land y$ and $x \lor y$ exist for all $x, y \in P$, then $P$ is a lattice.
• If $\land S$ and $\lor S$ 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 $I$
  – For any $x, y \in I$, $x \lor y = \max(x, y)$, $x \land y = \min(x, y)$
  – But $\lor I$ and $\land I$ do not exist
  – $I \cup \{+\infty, -\infty\}$ is a complete lattice
Top and Bottom

• Greatest element of P (if it exists) is top (T)
• Least element of P (if it exists) is bottom (⊥)
Connection Between $\leq$, $\land$, and $\lor$

• The following 3 properties are equivalent:
  - $x \leq y$
  - $x \lor y = y$
  - $x \land y = x$
Lattices as Algebraic Structures

• Have defined $\lor$ and $\land$ in terms of $\leq$
• Will now define $\leq$ in terms of $\lor$ and $\land$
  – Start with $\lor$ and $\land$ as arbitrary algebraic operations
    that satisfy associative, commutative, idempotence,
    and absorption laws
  – Will define $\leq$ using $\lor$ and $\land$
  – Will show that $\leq$ is a partial order
• Intuitive concept of $\lor$ and $\land$ as information
  combination operators (or, and)
Algebraic Properties of Lattices

Assume arbitrary operations $\lor$ and $\land$ such that

- $(x \lor y) \lor z = x \lor (y \lor z)$ (associativity of $\lor$)
- $(x \land y) \land z = x \land (y \land z)$ (associativity of $\land$)
- $x \lor y = y \lor x$ (commutativity of $\lor$)
- $x \land y = y \land x$ (commutativity of $\land$)
- $x \lor x = x$ (idempotence of $\lor$)
- $x \land x = x$ (idempotence of $\land$)
- $x \lor (x \land y) = x$ (absorption of $\lor$ over $\land$)
- $x \land (x \lor y) = x$ (absorption of $\land$ over $\lor$)
Connection Between $\land$ and $\lor$

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

\[ T = \{ \{ d1, d2, d3 \} \} \]

\[ \perp = \{ \} \]

\[ \{ d1, d2 \} \]

\[ \{ d1, d3 \} \]

\[ \{ d2 \} \]

\[ \{ d3 \} \]
Meet and Join Operations

\{ d_1, d_2 \} \land \{ d_2, d_3 \} = ???

\[ T = \{ d_1, d_2, d_3 \} \]

\[ \perp = \{ \} \]
Meet and Join Operations

\{ d1, d2 \} \land \{ d2, d3 \} = ???
Meet and Join Operations

\[
\{ \text{d1, d2} \} \land \{ \text{d2, d3} \} = ???
\]

\[
T = \{ \text{d1, d2, d3} \}
\]

\[
\perp = \{ \}\n\]
Meet and Join Operations

\[
\{ d1, d2 \} \land \{ d2, d3 \} = \{ d2 \}
\]
Meet and Join Operations

\{ d1, d2 \} \lor \{ d3 \} = ????

T = \{ d1, d2, d3 \}

\bot = \{ \}
Meet and Join Operations

\{ d1, d2 \} \lor \{ d3 \} = ???

T = \{ d1, d2, d3 \}

⊥ = \{ \}

\{ d1 \}

\{ d1, d2 \}

\{ d1, d3 \}

\{ d2 \}

\{ d2, d3 \}

\{ d3 \}
Meet and Join Operations

\[ \{ d_1, d_2 \} \lor \{ d_3 \} = ??? \]
Meet and Join Operations

\[ \{ d_1, d_2 \} \lor \{ d_3 \} = ??? \]
Meet and Join Operations

\{ d1, d2 \} \lor \{ d3 \} = \{ d1, d2, d3 \}
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 $\lor$ to combine values at control-flow join points
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
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:
  \[
  \forall f,g \in F. \text{ the function } h = \lambda x. f(g(x)) \in F
  \]
- Each $f \in F$ must be monotone:
  $x \leq y$ implies $f(x) \leq f(y)$
- Sometimes all $f \in F$ are distributive:
  \[
  f(x \lor y) = f(x) \lor f(y)
  \]
- Distributivity implies monotonicity
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
  – $\forall n. \ out_n = f_n(in_n)$
  – $\forall n \neq n_0. \ in_n = \lor \{ \ out_m . m \ in \ pred(n) \}$
  – $in_{n0} = I$
  – Where $I$ summarizes information at start of program
Dataflow Equations

- Compiler processes program to obtain a set of dataflow equations
  \[ \text{out}_n := f_n(\text{in}_n) \]
  \[ \text{in}_n := \lor \{ \text{out}_m \cdot m \text{ in } \text{pred}(n) \} \]
- Conceptually separates analysis problem from program
Worklist Algorithm for Solving Forward Dataflow Equations

for each \( n \) do
    \( \text{out}_n := f_n(\bot) \)
\( \text{in}_{n_0} := I; \)
\( \text{out}_{n_0} := f_{n_0}(I) \)
worklist := \( N - \{ n_0 \} \)
while worklist \( \neq \emptyset \) do
    remove a node \( n \) from worklist
    \( \text{in}_n := \lor \{ \text{out}_m . m \text{ in } \text{pred}(n) \} \)
    \( \text{out}_n := f_n(\text{in}_n) \)
    if \( \text{out}_n \) changed then
        worklist := worklist \( \cup \) succ(n)
Correctness Argument

• Why result satisfies dataflow equations

• Whenever process a node \( n \), set \( \text{out}_n := f_n(\text{in}_n) \)
  Algorithm ensures that \( \text{out}_n = f_n(\text{in}_n) \)

• Whenever \( \text{out}_m \) changes, put \( \text{succ}(m) \) on worklist. Consider any node \( n \in \text{succ}(m) \). It will eventually come off worklist and algorithm will set
  \[
  \text{in}_n := \lor \{ \text{out}_m \cdot m \in \text{pred}(n) \}
  \]
  to ensure that \( \text{in}_n = \lor \{ \text{out}_m \cdot m \in \text{pred}(n) \} \)

• 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)
- $v = \bigcup$ (order is $\subseteq$)
- $\bot = \emptyset$
- $I = \text{in}_{n_0} = \bot$
- $F =$ 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

\[ T = \{ \{d1, d2, d3\} \} \]

\[ \perp = \{\} \]
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
  - $\lambda x. a \cup (x - b) = \lambda x. x \in F$ (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

• $P = \text{powerset of set of all expressions in program (all subsets of set of expressions)}$
• $\vee = \cap$ (order is $\subseteq$)
• $\perp = P$
• $I = \text{in}_{n_0} = \emptyset$
• $F = \text{all functions } f \text{ of the form } f(x) = a \cup (x-b)$
  - $b$ is set of expressions that node kills
  - $a$ is set of expressions that node generates
• Another GEN/KILL analysis
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.
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
  – $\forall n. \ in_n = f_n(out_n)$
  – $\forall n \notin N_{final}. \ out_n = \lor \{ in_m \ . \ m \ in \ succ(n) \}$
  – $\forall n \in N_{final} = out_n = O$
  – Where O summarizes information at end of program
Worklist Algorithm for Solving Backward Dataflow Equations

for each \( n \) do
    \( \text{in}_n := f_n(\bot) \)
for each \( n \in N_{\text{final}} \) do
    \( \text{out}_n := O \)
    \( \text{in}_n := f_n(O) \)
worklist := \( N - N_{\text{final}} \)
while worklist \( \neq \emptyset \) do
    remove a node \( n \) from worklist
    \( \text{out}_n := \lor \{ \text{in}_m . m \in \text{succ}(n) \} \)
    \( \text{in}_n := f_n(\text{out}_n) \)
    if \( \text{in}_n \) changed then
        worklist := worklist \( \cup \) pred(\( n \))
Live Variables

- $P = \text{powerset of set of all variables in program}$
  (all subsets of set of variables in program)
- $\nu = \cup$ (order is $\subseteq$)
- $\bot = \emptyset$
- $O = \emptyset$
- $F = \text{all functions } f \text{ of the form } f(x) = a \cup (x - b)$
  - $b$ is set of variables that node kills
  - $a$ is set of variables that node reads
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 $n \leq e$, 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 $n$, 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
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)
Static Single Assignment (SSA) Form

\[
\begin{align*}
a &= 1 \\
b &= a + 2 \\
c &= a + b \\
a &= a + 1 \\
d &= a + b
\end{align*}
\]
Static Single Assignment (SSA) Form

\[
\begin{align*}
a &= 1 \\
b &= a + 2 \\
c &= a + b \\
a &= a + 1 \\
d &= a + b
\end{align*}
\]

\[
\begin{align*}
a_1 &= 1 \\
b_1 &= a_1 + 2 \\
c_1 &= a_1 + b_1 \\
a_2 &= a_1 + 1 \\
d_1 &= a_2 + b_1
\end{align*}
\]
Static Single Assignment (SSA) Form

\[
\begin{align*}
    a &= 1 \\
    c &= a + 2 \\
    b &= 1 \\
    c &= b + 2 \\
    d &= a + b + c
\end{align*}
\]
Static Single Assignment (SSA) Form

\[ a = 1 \]
\[ c = a + 2 \]

\[ b = 1 \]
\[ c = b + 2 \]

\[ d = a + b + c \]

\[ a_1 = 1 \]
\[ c_1 = a_1 + 2 \]

\[ b_1 = 1 \]
\[ c_2 = b_1 + 2 \]

\[ c_3 = \Phi(c_1, c_2) \]
\[ d_1 = c_3 + 2 \]
Summary

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