Foundations of Dataflow Analysis

Control Flow Graph:
- Nodes $N$ – statements of program
- Edges $E$ – flow of control
  - $\text{pred}(n)$ = set of all immediate predecessors of $n$
  - $\text{succ}(n)$ = set of all immediate successors of $n$
- Start node $s_0$
- Set of final nodes $N_{\text{final}}$

Terminology: Program Points

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

Terminology: Control-Flow Graph

Terminology: Extended Basic Block

EBB: Conceptually it is a program sequence with only one entry point but possibly several exit points.

Path of an EBB: A sequence of basic blocks $B_1, B_2, \ldots, B_n$ where all $B_i$ ($i > 1$) have a unique predecessor from the set $B_1, \ldots, B_{i-1}$.

Dataflow Analysis

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

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

Forward Dataflow Analysis

- Analysis propagates values forward through control flow graph with flow of control
  - Each node has a transfer function \( f \)
    - Input – value at program point before node
    - Output – new value at program point after node
  - Values flow from program points after predecessor nodes to program points before successor nodes
  - At join points, values are combined using a merge function
- Canonical Example: Reaching Definitions

Backward Dataflow Analysis

- Analysis propagates values backward through control flow graph against flow of control
  - Each node has a transfer function \( f \)
  - Input – value at program point after node
  - Output – new value at program point before node
  - Values flow from program points before successor nodes to program points after predecessor nodes
  - At split points, values are combined using a merge function
- Canonical Example: Live Variables

Partial Orders

- Set \( P \)
  - Partial order \( \leq \)
    - \( \forall x,y,z \in P \)
      - \( x \leq x \) (reflexive)
      - \( x \leq y \) and \( y \leq x \) implies \( x = y \) (antisymmetric)
      - \( x \leq y \) and \( y \leq z \) implies \( x \leq z \) (transitive)

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 \}\}
Coverings

- Notation: \( x < y \) if \( x \leq y \) and \( x \neq y \)
- \( x \) is covered by \( y \) (\( y \) covers \( x \)) if
  - \( x < y \), and
  - \( -x < y \) 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 \((x \text{ bitwise}_\text{and} y) = x\)

  ![Hasse Diagram](image)

We can visualize a partial order with a Hasse Diagram

- If \( y \) covers \( x \)
  - Line from \( y \) to \( x \)
  - \( y \) is above \( x \) in diagram

Lattices

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

Top and Bottom

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

Connection between \( \leq \), \( \wedge \), and \( \vee \)

The following 3 properties are equivalent:
- \( x \leq y \)
- \( x \lor y = y \)
- \( x \land y = x \)
- Will prove:
  - \( x \leq y \) implies \( x \lor y = y \) and \( x \land y = x \)
  - \( x \lor y = y \) implies \( x \leq y \)
  - \( x \land y = x \) implies \( x \leq y \)
- By Transitivity,
  - \( x \lor y = y \) implies \( x \land y = x \)
  - \( x \land y = x \) implies \( x \lor y = y \)

Connecting Lemma Proofs (1)

- Proof of \( x \leq y \) implies \( x \lor y = y \)
  - \( x \leq y \) implies \( y \) is an upper bound of \( \{x, y\} \).
  - Any upper bound \( z \) of \( \{x, y\} \) must satisfy \( y \leq z \).
  - So \( y \) is least upper bound of \( \{x, y\} \) and \( x \lor y = y \)
- Proof of \( x \leq y \) implies \( x \land y = x \)
  - \( x \leq y \) implies \( x \) is a lower bound of \( \{x, y\} \).
  - Any lower bound \( z \) of \( \{x, y\} \) must satisfy \( z \leq x \).
  - So \( x \) is greatest lower bound of \( \{x, y\} \) and \( x \land y = x \)
Connecting Lemma Proofs (2)

• Proof of \( x \lor y = y \) implies \( x \leq y \)
  – \( y \) is an upper bound of \( \{x, y\} \) implies \( x \leq y \)

• Proof of \( x \land y = x \) implies \( x \leq y \)
  – \( x \) is a lower bound of \( \{x, y\} \) implies \( x \leq y \)

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

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 \( \land \))
– \( x \lor (x \land y) = x \) (absorption of \( \lor \) over \( \land \))
– \( x \land (x \lor y) = x \) (absorption of \( \land \) over \( \lor \))

Connection Between \( \land \) and \( \lor \)

Theorem: \( x \lor y = y \) 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 \( y = x \lor y \)
  \( y = y \lor (x \land y) \) (by commutativity)
  \( = y \lor x \) (by assumption)
  \( = x \lor y \) (by commutativity)

Properties of \( \leq \)

• Define \( x \leq y \) if \( x \lor y = y \)

• Proof of transitive property. Must show that
  \( x \lor y = y \) and \( y \lor z = z \) implies \( x \lor z = z \)
  \( x \lor z = x \lor (y \lor z) \) (by assumption)
  \( = (x \lor y) \lor z \) (by associativity)
  \( = y \lor z \) (by assumption)
  \( = z \) (by assumption)

• Proof of antisymmetry property. Must show that
  \( x \lor y = y \) and \( y \lor x = x \) implies \( x = y \)
  \( x = y \lor x \) (by assumption)
  \( = x \lor y \) (by commutativity)
  \( = y \) (by assumption)

• Proof of reflexivity property. Must show that
  \( x \lor x = x \)
  \( x \lor x = x \) (by idempotence)

Properties of \( \leq \)

• Proof of antisymmetry property. Must show that
  \( x \lor y = y \) and \( y \lor x = x \) implies \( x = y \)
  \( x = y \lor x \) (by assumption)
  \( = x \lor y \) (by commutativity)
  \( = y \) (by assumption)

• Proof of reflexivity property. Must show that
  \( x \lor x = x \)
  \( x \lor x = x \) (by idempotence)
Properties of $\leq$

- Induced operation $\leq$ agrees with original definitions of $\lor$ and $\land$, i.e.,
  \[ x \lor y = \sup \{x, y\} \]
  \[ x \land y = \inf \{x, y\} \]

Proof of $x \lor y = \sup \{x, y\}$

- Consider any upper bound $u$ for $x$ and $y$.
- Given $x \lor u = u$ and $y \lor u = u$, must show $x \lor y \leq u$, i.e., $(x \lor y) \lor u = u$.
  \[ u = x \lor u \quad \text{(by assumption)} \]
  \[ = x \lor (y \lor u) \quad \text{(by assumption)} \]
  \[ = (x \lor y) \lor u \quad \text{(by associativity)} \]

Proof of $x \land y = \inf \{x, y\}$

- Consider any lower bound $l$ for $x$ and $y$.
- Given $x \land l = l$ and $y \land l = l$, must show $l \leq x \land y$, i.e., $(x \land y) \land l = l$.
  \[ 1 = x \land 1 \quad (\text{by assumption}) \]
  \[ = x \land (y \land l) \quad (\text{by assumption}) \]
  \[ = (x \land y) \land l \quad (\text{by associativity}) \]

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.
- $P$ satisfies the ascending chain condition if for all sequences $x_1 \leq x_2 \leq \ldots$ there exists $n$ such that $x_n = x_{n+1} = \ldots$

Transfer Functions

- Assume a lattice of abstract values $P$.
- Transfer function $f : P \to P$ for each node in control flow graph.
- $f$ models effect of the node on program information.

Properties of Transfer Functions

Each dataflow analysis problem has a set $F$ of transfer functions $f : P \to P$.
- 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: $f(x \lor y) = f(x) \lor f(y)$.
- Distributivity implies monotonicity.
Distributivity Implies Monotonicity

Proof:
• Assume \( f(x \lor y) = f(x) \lor f(y) \)
• Must show: \( x \lor y = y \) implies \( f(x) \lor f(y) = f(y) \)
  \[
  f(y) = f(x \lor y) \quad \text{(by assumption)}
  = f(x) \lor f(y) \quad \text{(by distributivity)}
  \]

Forward Dataflow Analysis

• Simulates execution of program forward with flow of control
• For each node \( n \), have
  - \( \text{in}_n \) – value at program point before \( n \)
  - \( \text{out}_n \) – value at program point after \( n \)
  - \( f_n \) – transfer function for \( n \) (given \( \text{in}_n \) computes \( \text{out}_n \))
• Require that solutions satisfy
  - \( \forall n, \text{out}_n = f_n(\text{in}_n) \)
  - \( \forall n \neq n_0, \text{in}_n = \lor \{ \text{out}_m | m \in \text{pred}(n) \} \)
  - \( \text{in}_{n_0} = \bot \)

Dataflow Equations

• Result is a set of dataflow equations
  \[
  \text{out}_n := f_n(\text{in}_n) \\
  \text{in}_n := \lor \{ \text{out}_m | m \in \text{pred}(n) \}
  \]
• Conceptually separates analysis problem from program

Worklist Algorithm for Solving Forward Dataflow Equations

for each \( n \) do \( \text{out}_n := f_n(\bot) \)
worklist := \( N \)
while worklist \( \neq \emptyset \) do
  remove a node \( n \) from worklist
  \( \text{inn} := \lor \{ \text{out}_m | m \in \text{pred}(n) \} \)
  \( \text{out}_n := f_n(\text{inn}) \)
  if \( \text{out}_n \) changed then
    worklist := worklist \cup \text{succ}(n)

Correctness Argument

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

Termination Argument

Why does the algorithm terminate?
• Sequence of values taken on by \( \text{in}_n \) or \( \text{out}_n \) is a chain. If values stop increasing, the worklist empties and the algorithm terminates.
• If the lattice has the ascending chain property, the algorithm terminates
  – Algorithm terminates for finite lattices
  – For lattices without the ascending chain property, we must use a widening operator
Widening Operators

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

Reaching Definitions

- Concept of definition and use
  - \( z = x \times y \)
  - is a definition of \( z \)
  - is a use of \( x \) and \( y \)
- A definition reaches a use if
  - the value written by definition
  - may be read by the use.

Reaching Definitions Framework

- \( P = \) powerset of set of all definitions in program (all subsets of set of definitions in program)
- \( \lor = \) \( \cup \) (order is \( \subseteq \))
- \( \bot = \) \( \emptyset \)
- \( F = \) 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
  - \( f(x) = \text{GEN} \cup (x\text{-KILL}) \)

Does Reaching Definitions Framework Satisfy Properties?

- \( \subseteq \) satisfies conditions for \( \leq \)
  - \( x \leq y \) and \( y \leq z \) implies \( x \leq z \) (transitivity)
  - \( x \leq y \) and \( y \leq x \) implies \( y = x \) (antisymmetry)
  - \( x \leq x \) (reflexivity)
- \( F \) satisfies transfer function conditions
  - \( \lambda x. (x \lor (x\cdot F)) = \lambda x. x \in F \) (identity)
  - Will show \( f(x \lor y) = f(x) \lor f(y) \) (distributivity)
  - \( f(x \lor y) = (a \lor (x \cdot b)) \lor (a \lor (y \cdot b)) \)
  - \( a \lor (x \cdot b) \lor (y \cdot b) \)
  - \( a \lor ((x \cup y) \cdot b) \)
  - \( f(x \cup y) \)

Does Reaching Definitions Framework Satisfy Properties?

- What about composition?
  - Given \( f_1(x) = a_1 \cup (x \cdot b_1) \) and \( f_2(x) = a_2 \cup (x \cdot b_2) \)
  - Must show \( f_1(f_2(x)) \) can be expressed as \( a \cup (x \cdot b) \)
  - \( f_1(f_2(x)) = a_1 \cup ((a_2 \cup (x \cdot b_2)) \cdot b_1) \)
  - \( = a_1 \cup ((a_2 \cdot b_2) \cup ((x \cdot b_2) \cdot b_1)) \)
  - \( = (a_1 \cdot (a_2 \cdot b_2)) \cup ((x \cdot b_2) \cdot b_1)) \)
  - \( = (a_1 \cdot (a_2 \cdot b_2)) \cup ((x \cdot b_2) \cdot b_1)) \)
  - Let \( a = (a_1 \cdot (a_2 \cdot b_2)) \) and \( b = b_2 \cup b_1 \)
  - Then \( f_1(f_2(x)) = a \cup (x \cdot b) \)
General Result
All GEN/KILL transfer function frameworks satisfy the properties:
– Identity
– Distributivity
– Compositionality

Available Expressions Framework
• P = powerset of set of all expressions in program (all subsets of set of expressions)
• \( \lor = \cap \) (order is \( \supseteq \))
• \( \bot = P \) (but \( \text{in}_{ \emptyset } = \emptyset \))
• 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

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 the way the results of the analysis are to be used.

Backward Dataflow Analysis
• Simulates execution of program backward against the flow of control
• For each node \( n \), we 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 solutions satisfy
  – \( \forall n. \text{in}_{ n } = f_{ n } (\text{out}_{ n }) \)
  – \( \forall n \notin N_{\text{final}}. \text{out}_{ n } = \lor \{ \text{in}_{ m } | m \in \text{succ}(n) \} \)
  – \( \forall n \in N_{\text{final}} = \text{out}_{ n } = \bot \)

Worklist Algorithm for Solving Backward Dataflow Equations
for each \( n \) do \( \text{in}_n := f_n(\bot) \)
worklist := N
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 Analysis Framework
• P = powerset of set of all variables in program (all subsets of set of variables in program)
• \( \lor = \cup \) (order is \( \subseteq \))
• \( \bot = \emptyset \)
• F = all functions \( f \) of the form \( f(x) = a \cup (x-b) \)
  – b is set of variables that the node kills
  – a is set of variables that the node reads
Meaning of Dataflow Results

• Connection between executions of program and dataflow analysis results
• Each execution generates a trajectory of states: \( s_0; s_1; \ldots; s_k \), where each \( s_i \in ST \)
• Map current state \( s_k \) to
  – Program point \( n \) where execution located
  – Value \( x \) in dataflow lattice
• Require \( x \leq \text{inn} \)

Abstraction Function for Forward Dataflow Analysis

• Meaning of analysis results is given by an abstraction function \( AF: ST \to P \)
• Require that for all states \( s \)
  \[ AF(s) \leq \text{inn}_n \]
  where \( n \) is program point where the execution is located in state \( s \), and \( \text{inn}_n \) is the abstract value before that point.

Sign Analysis Example

Sign analysis - compute sign of each variable \( v \)
• Base Lattice: flat lattice on \{-, zero, +\}
  - TOP
  - zero
  - +
  - BOT
• Actual lattice records a value for each variable
  – Example element: \([a \to +, b \to \text{zero}, c \to -]\)

Interpretation of Lattice Values

If value of \( v \) in lattice is:
  – BOT: no information about the sign of \( v \)
  – -: variable \( v \) is negative
  – zero: variable \( v \) is 0
  – +: variable \( v \) is positive
  – TOP: \( v \) may be positive or negative or 0

Operation \( \otimes \) on Lattice

<table>
<thead>
<tr>
<th>( \otimes )</th>
<th>BOT</th>
<th>-</th>
<th>zero</th>
<th>+</th>
<th>TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOT</td>
<td>BOT</td>
<td>-</td>
<td>zero</td>
<td>+</td>
<td>TOP</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>+</td>
<td>zero</td>
<td>-</td>
<td>TOP</td>
</tr>
<tr>
<td>zero</td>
<td>zero</td>
<td>zero</td>
<td>zero</td>
<td>zero</td>
<td>zero</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>-</td>
<td>zero</td>
<td>+</td>
<td>TOP</td>
</tr>
<tr>
<td>TOP</td>
<td>TOP</td>
<td>TOP</td>
<td>zero</td>
<td>TOP</td>
<td>TOP</td>
</tr>
</tbody>
</table>

Transfer Functions

Defined by structural induction on the shape of nodes:
  – If \( n \) of the form \( v = c \)
    • \( f(x) = x[v \to +] \) if \( c \) is positive
    • \( f(x) = x[v \to \text{zero}] \) if \( c \) is 0
    • \( f(x) = x[v \to -] \) if \( c \) is negative
  – If \( n \) of the form \( v_i = v_j \)
    • \( f(x) = x[v_i \to x[v_j \otimes x[v_j]]] \)
Abstraction Function

• $AF(s)[v] = \text{sign of } v$
  – $AF([a\to 5, b\to 0, c\to 2]) = [a\to +, b\to \text{zero}, c\to -]$

• Establishes meaning of the analysis results
  – If analysis says a variable $v$ has a given sign
    – then $v$ always has that sign in actual execution.

• Two sources of imprecision
  – Abstraction Imprecision – concrete values (integers) abstracted as lattice values (-, zero, and +)
  – Control Flow Imprecision – one lattice value for all different possible flow of control possibilities

Imprecision Example

Abstraction Imprecision:

$$a = 1$$

$$[a\to+]$$

$$b = -1$$

$$[a\to+, b\to+]$$

$$c = a^b$$

Control Flow Imprecision:

$$[b\to \text{TOP}]$$ summarizes results of all executions.

In any execution state $s$, $AF(s)[b] \neq \text{TOP}$

General Sources of Imprecision

• Abstraction Imprecision
  – Lattice values less precise than execution values
  – Abstraction function throws away information

• Control Flow Imprecision
  – Analysis result has a single lattice value to summarize results of multiple concrete executions
  – Join operation $\lor$ moves up in lattice to combine values from different execution paths

  Typically if $x \leq y$, then $x$ is more precise than $y$.

Why Have Imprecision?

ANSWER: To make analysis tractable

• Conceptually infinite sets of values in execution
  – Typically abstracted by finite set of lattice values

• Execution may visit infinite set of states
  – Abstracted by computing joins of different paths

Augmented Execution States

• Abstraction functions for some analyses require augmented execution states
  – Reaching definitions: states are augmented with the definition that created each value
  – Available expressions: states are augmented with expression for each value

Meet Over All Paths Solution

• What solution would be ideal for a forward dataflow analysis problem?

• Consider a path $p = n_0, n_1, \ldots, n_k$ to a node $n$ (note that for all $i$, $n_i \in \text{pred}(n_{i+1})$)

• The solution must take this path into account:
  $$f_p(L) = (f_{n_k}(f_{n_{k-1}}(\ldots(f_{n_1}(\bot)) \ldots)) \leq n_k$$

• So the solution must have the property that
  $$\forall \{\mu(L) \mid p \text{ is a path to } n\} \leq n_k$$

  and ideally
  $$\forall \{\mu(L) \mid p \text{ is a path to } n\} = n_k$$
Soundness Proof of Analysis Algorithm

Property to prove:
For all paths p to n, \( f_p(\bot) \leq i_n \)

- Proof is by induction on the length of p
  - Uses monotonicity of transfer functions
  - Uses following lemma

Lemma:
The worklist algorithm produces a solution such that
if \( n \in \text{pred}(m) \) then \( o_n \leq i_m \)

Proof

- Base case: p is of length 0
  - Then \( p = n_0 \) and \( f_p(\bot) = \bot = i_{n_0} \)
- Induction step:
  - Assume theorem for all paths of length k
  - Show for an arbitrary path p of length k+1.

Induction Step Proof

- \( p = n_0, \ldots, n_k, n \)
- Must show \( (f_k(f_{k-1}(\ldots f_{n_1}(f_{n_0}(\bot)) \ldots))) \leq i_n \)
  - By induction, \( (f_k(f_{k-1}(\ldots f_{n_1}(f_{n_0}(\bot)) \ldots))) \leq i_{n_k} \)
  - By monotonicity, we get:
    - By lemma, \( o_{n_k} \leq i_n \)
    - By transitivity, \( (f_k(f_{k-1}(\ldots f_{n_1}(f_{n_0}(\bot)) \ldots))) \leq i_n \)

Distributivity

- Distributivity preserves precision
- If framework is distributive, then the worklist algorithm produces the meet over paths solution
  - For all n:
    \[
    \lor \{f_p(\bot) \mid p \text{ is a path to } n\} = i_n
    \]

Lack of Distributivity Example

Integer Constant Propagation (ICP)
- Flat lattice on integers

\[
\begin{array}{c|c|c|c|c|c}
  & \text{TOP} & -1 & 0 & 1 & \text{BOT} \\
\hline
-2 & \text{BOT} & -1 & 0 & 1 & \text{TOP} \\
\end{array}
\]
- Example element: \([a \rightarrow 3, b \rightarrow 2, c \rightarrow 5]\)

Transfer Functions

- If n of the form \( v = c \)
  - \( f_v(x) = x[v\rightarrow c] \)
- If n of the form \( v_1 = v_2 + v_3 \)
  - \( f_v(x) = x[v_1\rightarrow x[v_2] + x[v_3]] \)
- Lack of distributivity of ICP
  - Consider transfer function \( f \) for \( c = a + b \)
    - \( f([a \rightarrow 3, b \rightarrow 2]) \lor f([a \rightarrow 2, b \rightarrow 3]) = [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}, c \rightarrow 5] \)
    - \( f([a \rightarrow 3, b \rightarrow 2], [a \rightarrow 2, b \rightarrow 3]) = f([a \rightarrow \text{TOP}, b \rightarrow \text{TOP}]) = [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}, c \rightarrow \text{TOP}] \)
Lack of Distributivity Anomaly

\[ \begin{align*}
    a &= 2 & a &= 3 \\
    b &= 3 & b &= 2 \\
    [a \rightarrow 2, b \rightarrow 3] & \quad [a \rightarrow 3, b \rightarrow 2] \\
    [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}] & \quad [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}] \\
    c &= a + b & \text{Lack of Distributivity Imprecision:} & \quad [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}, c \rightarrow 5] \text{ more precise} \\
    [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}, c \rightarrow \text{TOP}] & \quad [a \rightarrow \text{TOP}, b \rightarrow \text{TOP}, c \rightarrow \text{TOP}] 
\end{align*} \]

Summary

- Formal dataflow analysis framework
  - Lattices, partial orders
  - Transfer functions, joins and splits
  - Dataflow equations and fixed point solutions
- Connection with program
  - Abstraction function AF: S \rightarrow P
  - For any state s and program point n, AF(s) \leq \text{in}_n
  - Meet over paths solutions, distributivity