Dataflow Analysis (Continued)

CIS410/510 Program Analysis and Transformation

Lattices

- **Lattice:**
  - Set augmented with a partial order relation $\sqsubseteq$
  - Each subset has a LUB and a GLB
  - Can define: meet $\sqcap$, join $\sqcup$, top $\top$, bottom $\bot$

- **Use lattice** to express information about a point in a program, where

  $S_1 \sqsubseteq S_2$ means “$S_1$ is less or equally precise as $S_2$”

- **To compute information:** build constraints that describe how the lattice information changes
  - Effect of instructions: transfer functions
  - Effect of control flow: meet operation
Transfer functions

- Let $L = \text{dataflow information lattice}$

- **Transfer function $F_I : L \rightarrow L$** for each instruction $I$
  - Describes how $I$ modifies the information in the lattice
  - If $\text{in}[I]$ is info before $I$ and $\text{out}[I]$ is info after $I$, then
    \begin{align*}
    \text{Forward analysis:} & \quad \text{out}[I] = F_I(\text{in}[I]) \\
    \text{Backward analysis:} & \quad \text{in}[I] = F_I(\text{out}[I])
    \end{align*}

- **Transfer function $F_B : L \rightarrow L$** for each basic block $B$
  - Is composition of transfer functions of instructions in $B$
  - If $\text{in}[B]$ is info before $B$ and $\text{out}[B]$ is info after $B$, then
    \begin{align*}
    \text{Forward analysis:} & \quad \text{out}[B] = F_B(\text{in}[B]) \\
    \text{Backward analysis:} & \quad \text{in}[B] = F_B(\text{out}[B])
    \end{align*}

Control flow

- **Meet operation** models how to combine information at split/join points in the control flow
  - If $\text{in}[B]$ is info before $B$ and $\text{out}[B]$ is info after $B$, then:
    \begin{align*}
    \text{Forward analysis:} & \quad \text{in}[B] = \bigcap \{\text{out}[B'] \mid B' \in \text{pred}(B)\} \\
    \text{Backward analysis:} & \quad \text{out}[B] = \bigcap \{\text{in}[B'] \mid B' \in \text{succ}(B)\}
    \end{align*}

- Can alternatively use join operation $\bigcup$ (equivalent to using the meet operation $\bigcap$ in the reversed lattice)
Treatment as $F : L^n \to L^n$

- For a data flow analysis problem
  - With lattice $L$
  - Basic blocks $B_1, \ldots, B_n$
  - Transfer functions $F_1, \ldots, F_n$

- Treat as
  - Iteration of function $F : L^n \to L^n$
    $\tau, F(\tau), F(F(\tau)), \ldots$
  - Where $F$ summarizes effect of one sweep for all blocks $B$ in a given order of either
    $\text{out}[B] = \ldots$ and $\text{in}[B] = F_B(\text{out}[B])$ (for backward)
    $\text{in}[B] = \ldots$ and $\text{out}[B] = F_B(\text{in}[B])$ (for forward)

Monotonicity

- Function $F : L \to L$ is monotonic if
  $x \leq y$ implies $F(x) \leq F(y)$

- A monotonic function is “order preserving”

- Contrast with
  $\text{For all } x, \quad F(x) \nleq x$

- $F$ is monotonic but $C = F(B) \nleq B$
Monotonicity of \textit{meet}

- Meet operation is monotonic over \(L \times L\), i.e.,
  \[ x_1 \trianglelefteq y_1 \text{ and } x_2 \trianglelefteq y_2 \text{ implies } (x_1 \land x_2) \trianglelefteq (y_1 \land y_2) \]

- \textbf{Proof}:
  - any lower bound of \(\{x_1, x_2\}\) is also a lower bound of \(\{y_1, y_2\}\), because \(x_1 \trianglelefteq y_1\) and \(x_2 \trianglelefteq y_2\)
  - \(x_1 \land x_2\) is a lower bound of \(\{x_1, x_2\}\)
  - So \(x_1 \land x_2\) is a lower bound of \(\{y_1, y_2\}\)
  - But \(y_1 \land y_2\) is the greatest lower bound of \(\{y_1, y_2\}\)
  - Hence \((x_1 \land x_2) \trianglelefteq (y_1 \land y_2)\)

\textbf{Fixed points}

- \(x\) in lattice \(L\) is a \textbf{fixed point of function} \(F\) \textbf{iff} \(x = F(x)\)
- \textbf{Tarski-Knaster Fixed Point Theorem}. The fixed points of a monotonic function on a complete lattice form a complete lattice. In particular, there is a maximal fixed point (MFP).
Chains in lattices

- A chain in a lattice L is a totally ordered subset S of L: $x \leq y$ or $y \leq x$ for any $x, y \in S$

- In other words:
  Elements in a totally ordered subset S can be indexed to form an ascending sequence:
  $x_1 \leq x_2 \leq x_3 \leq ...$
  or they can be indexed to form a descending sequence:
  $x_1 \geq x_2 \geq x_3 \geq ...$

- Height of a lattice = size of its largest chain
- Lattice with finite height: only has finite chains

Iterative computation of solution

- Let $F$ be a monotonic function over lattice L
- $\top \geq F(\top) \geq F(F(\top)) \geq ...$ is a descending chain
- If L has finite height, the chain ends at the maximal fixed point of $F$ (MFP)
Multiple solutions

- Dataflow equations may have multiple solutions

- Example: live variables

  Equations:
  \[
  \begin{align*}
  L_1 &= L_2 - \{y\} \\
  L_3 &= (L_4 - \{x\}) \cup \{y\} \\
  L_2 &= L_1 \cup L_3 \\
  L_4 &= \{x\}
  \end{align*}
  \]

  Could compute either \(L_2\) or \(L_3\) first.

  Solution 1: \(L_1 = \emptyset, \ L_2 = \{y\}, \ L_3 = \{y\}, \ L_4 = \{x\}\)

  Solution 2: \(L_1 = \{x\}, \ L_2 = \{x, y\}, \ L_3 = \{y\}, \ L_4 = \{x\}\)

For any solution \(FP\) of the dataflow equations \(FP \sqsubseteq MFP \ FP\) is said to be a conservative or safe solution

Meet over paths solution (forward)

- Is MFP the best solution to an analysis problem?

- Alternative to MFP: a different way to compute solution
  - Let \(G\) be the control flow graph with start block \(B_0\)
  - For each path \(p_n = [B_0, B_1, \ldots, B_n]\) from \(B_0\) to block \(B_n\)
    define \(F[p_n] = F_{B_{n-1}} \circ F_{B_{n-1}} \circ \ldots \circ F_{B_0}\)
  - Compute solution as
    \[
    \text{in}[B_n] = \prod \{ F[p_n](\text{start value}) | \text{all paths } p_n \text{ from } B_0 \text{ to } B_n \}\]

- This solution is the Meet Over Paths (MOP) solution for
MFP versus MOP

- **Precision:** MOP solution is at least as precise as MFP
  \[ \text{MFP} \subseteq \text{MOP} \]

- **Why not use MOP?**
  - MOP is intractable in practice
    1. Exponential number of paths: for a program consisting of a sequence of N if statement, there will \(2^N\) paths in the control flow graph
    2. Infinite number of paths: for loops in the CFG

Distributivity

- Function \(F : L \rightarrow L\) is **distributive** if
  \[ F(x \sqcap y) = F(x) \sqcap F(y) \]

- **Property:** \(F\) is monotonic iff \(F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y)\)
  - any distributive function is monotonic!
Importance of Distributivity

- **Property:** if transfer functions are *distributive*, then the solution to the dataflow equations is identical to the meet-over-paths solution

\[ \text{MFP} = \text{MOP} \]

- For distributive transfer functions, can compute the intractable MOP solution using the iterative fixed-point algorithm

**Better Than MOP?**

- Is MOP the best solution to the analysis problem?
- MOP computes solution for all paths in the CFG
- There may be paths that will never occur in any execution
- So MOP is conservative

- **IDEAL** = solution that takes into account only paths that occur in some execution
- This is the best solution... but it is undecidable

\[ \begin{align*}
  &\text{if } (c) \\
  &x = 1 \quad x = 2 \\
  &\text{if } (c) \\
  &y = y + 2 \quad y = x + 1
\end{align*} \]
Dataflow Equations

- Solve equations: use an iterative algorithm
  - Initialize in[B_3] = start value
  - Initialize everything else to T
  - Repeatedly apply rules
  - Stop when reach a fixed point

Kildall Algorithm (forward)

\[
\text{in}[B_3] = \text{start value} \\
\text{out}[B] = T, \text{ for all } B
\]

**repeat**
  **for** each basic block \( B \neq B_3 \)
  \[
  \text{in}[B] = \prod \{ \text{out}[B'] \mid B' \subseteq \text{pred}(B) \}
  \]
  **for** each basic block \( B \)
  \[
  \text{out}[B] = F_B(\text{in}[B])
  \]
  **until** no change
Efficiency

- **Algorithm is inefficient**
  - Effects of basic blocks re-evaluated even if the input information has not changed

- **Better**: re-evaluate blocks only when necessary

- **Use a worklist algorithm**
  - Keep of list of blocks to evaluate
  - Initialize list to the set of all basic blocks
  - If out[B] changes after evaluating out[B] = F_B(in[B]), then add all successors of B to the list

---

**Worklist Algorithm (forward)**

in[B_0] = start value  
out[B] = ⊤, for all B  
worklist = set of all basic blocks B

repeat  
  remove a node B from the worklist  
in[B] = \prod \{out[B'] \mid B' ∈ \text{pred}(B)\} out[B]  
= F_B(in[B])  
if out[B] has changed then  
  worklist = worklist ∪ succ(B)

until worklist = ∅
Correctness

- Initial algorithm is correct
  - If dataflow information does not change in the last iteration, then it satisfies the equations

- Worklist algorithm is correct
  - Maintains the invariant that
    \[
    \text{in}[B] = \bigcap \{\text{out}[B'] \mid B' \in \text{pred}(B)\}
    \]
    \[
    \text{out}[B] = F_B(\text{in}[B])
    \]
   for all the blocks B not in the worklist
  - At the end, worklist is empty

Summary

- Dataflow analysis
  - sets up system of equations
  - iteratively computes MFP
  - Terminates because transfer functions are monotonic and lattice has finite height

- Other possible solutions: FP, MOP, IDEAL
  All are safe solutions, but some are more precise:
  \[
  \text{FP} \subseteq \text{MFP} \subseteq \text{MOP} \subseteq \text{IDEAL}
  \]

- MFP = MOP if distributive transfer functions

- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP
Dataflow Analysis Instances

• Apply dataflow framework to several analysis problems:
  – Live variable analysis
  – Available expressions
  – Reaching definitions
  – Constant folding

• Discuss:
  – Implementation issues
  – Classification of dataflow analyses
Problem 1: Live Variables

- Compute live variables at each program point
- Live variable = variable whose value may be used later, in some execution of the program

- Dataflow information: sets of live variables
- Example: variables \{x,z\} may be live at program point p
- Is a backward analysis

- Let \(V = \) set of all variables in the program
- Lattice \((L, \subseteq)\), where:
  - \(L = 2^V\) (power set of \(V\), i.e., set of all subsets of \(V\))
  - Partial order \(\subseteq\) is set inclusion: \(\supseteq\)
    \[ S_1 \subseteq S_2 \iff S_1 \supseteq S_2 \]

LV: The Lattice

- Consider set of variables \(V = \{x,y,z\}\)
- Partial order: \(\supseteq\)
- Set \(V\) is finite implies lattice has finite height

- Meet operator: \(\cup\)
  (set union: out\([B]\) is union of in\([B']\), for all \(B' \subseteq \text{succ}(B)\))

- Top element: \(\emptyset\)
  (empty set)

- Smaller sets of live variables = more precise analysis
- All variables may be live = least precise
LV: Dataflow Equations

- Equations:
  \[
  \text{in}[B] = F_B(\text{out}[B]), \text{ for all } B
  \]
  \[
  \text{out}[B] = \bigcup\{\text{in}[B'] \mid B' \subseteq \text{succ}(B)\}, \text{ for all } B
  \]
  \[
  \text{out}[B_e] = X_0
  \]

- Meaning of union meet operator:
  “A variable is live at the end of a basic block B if it is live at the beginning of one of its successor blocks”

LV: Transfer Functions

- Transfer functions for basic blocks are composition of transfer functions of instructions in the block
- Define transfer functions for instructions

- General form of transfer functions:
  \[F_I(X) = (X - \text{def}[I]) \cup \text{use}[I]\]
  where:
  \[
  \text{def}[I] = \text{set of variables defined (written) by } I
  \]
  \[
  \text{use}[I] = \text{set of variables used (read) by } I
  \]

- Meaning of transfer functions:
  “Variables live before instruction I include: (1) variables live after I, but not written by I, and (2) variables used by I”
LV: Transfer Functions

- Define def/use for each type of instruction
  
  if I is $x = y \ \text{OP} \ z$ : 
  
  use[I] = \{y, z\} 
  
  def[I] = \{x\}
  
  if I is $x = \text{OP} \ y$ : 
  
  use[I] = \{y\} 
  
  def[I] = \{x\}
  
  if I is $x = y$ : 
  
  use[I] = \{y\} 
  
  def[I] = \{x\}
  
  if I is $x = \text{addr} \ y$ : 
  
  use[I] = \{} 
  
  def[I] = \{x\}
  
  if I is $\text{if} (x)$ : 
  
  use[I] = \{x\} 
  
  def[I] = \{} 
  
  if I is $\text{return} \ x$ : 
  
  use[I] = \{x\} 
  
  def[I] = \{} 
  
  if I is $x = f(y_1, \ldots, y_n)$ : 
  
  use[I] = \{y_1, \ldots, y_n\} 
  
  def[I] = \{x\}

- Transfer functions $F_i(X) = (X - \text{def}[I]) \cup \text{use}[I]$

- For each $F_i$, def[I] and use[I] are constants: they don’t depend on input information $X$

LV: Distributivity

- Are transfer functions: $F_i(X) = (X - \text{def}[I]) \cup \text{use}[I]$ distributive?

- Since def[I] is constant: $X - \text{def}[I]$ is distributive: $(X_1 \cup X_2) - \text{def}[I] = (X_1 - \text{def}[I]) \cup (X_2 - \text{def}[I])$ because: $(a \cup b) - c = (a - c) \cup (b - c)$

- Since use[I] is constant: $Y \cup \text{use}[I]$ is distributive: $(Y_1 \cup Y_2) \cup \text{use}[I] = (Y_1 \cup \text{use}[I]) \cup (Y_2 \cap \text{use}[I])$ because: $(a \cup b) \cup c = (a \cup c) \cup (b \cup c)$

- Put pieces together: $F_i(X)$ is distributive $F_i(X_1 \cup X_2) = F_i(X_1) \cup F_i(X_2)$
Live variables: Summary

- Lattice: \((2^V, \supseteq)\); has finite height
- Meet is set union, top is empty set
- Is a backward dataflow analysis
- Dataflow equations:
  \[ \text{in}[B] = F_B(\text{out}[B]), \text{for all } B \]
  \[ \text{out}[B] = \bigcup\{\text{in}[B'] | B' \in \text{succ}(B)\}, \text{for all } B \]
  \[ \text{out}[B_0] = X_0 \]
- Transfer functions: \(F_i(X) = (X - \text{def}[I]) \cup \text{use}[I]\)
  - are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution

Problem 2: Available expressions

- Compute available expressions at each program point
- Available expression = expression evaluated in all program executions, and its value would be the same if re-evaluated
- Is similar to available copies for constant propagation
- Dataflow information: sets of available expressions
- Example: expressions \(\{x+y, y-z\}\) are available at point \(p\)
- Is a forward analysis
- Let \(E = \text{set of all expressions in the program}\)
- Lattice \((L, \sqsubseteq)\), where:
  - \(L = 2^E\) (power set of \(E\), i.e., set of all subsets of \(E\))
  - Partial order \(\sqsubseteq\) is set inclusion: \(\supseteq\)
    \[ S_1 \sqsubseteq S_2 \text{ iff } S_1 \supseteq S_2 \]
AE: The lattice

- Consider set of expressions = \{x*z, x+y, y-z\}
- Denote e = x*z, f=x+y, g=y-z

- Partial order: \( \subseteq \)
- Set E is finite implies lattice has finite height

- Meet operator: \( \cap \)
  (set intersection)

- Top element: \{e,f,g\} (set of all expressions)

- Larger sets of available expressions = more precise analysis
- No available expressions = least precise

---

AE: Dataflow equations

- Equations:
  \[
  \begin{align*}
  \text{out}[B] &= F_B(\text{in}[B]), \text{ for all } B \\
  \text{in}[B] &= \cap \{\text{out}[B'] \mid B' \subseteq \text{pred}(B)\}, \text{ for all } B \\
  \text{in}[B_s] &= X_0 
  \end{align*}
  \]

- Meaning of intersection meet operator:
  “An expression is available at entry of block B if it is available at exit of all predecessor nodes”
AE: Transfer functions

- Define transfer functions for instructions

- General form of transfer functions:
  \[ F_i(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]
  where:
  - \text{kill}[I] = expressions “killed” by \( I \)
  - \text{gen}[I] = new expressions “generated” by \( I \)

- **Note:** this kind of transfer function is typical for many dataflow analyses!

- Meaning of transfer functions: “Expressions available after instruction \( I \) include: (1) expressions available before \( I \), but not killed by \( I \), and (2) expressions generated by \( I \)”

---

AE: Transfer functions

- Define kill/gen for each type of instruction

  - if \( I \) is \( x = y \text{ OP } z \):
    - \( \text{gen}[I] = \{ y \text{ OP } z \} \)
    - \( \text{kill}[I] = \{ E \mid x \in E \} \)
  
  - if \( I \) is \( x = \text{ OP } y \):
    - \( \text{gen}[I] = \{ \text{OP } z \} \) if \( I \)
    - \( \text{kill}[I] = \{ E \mid x \in E \} \)

  - if \( I \) is \( x = \text{y} \):
    - \( \text{gen}[I] = \{ \} \)
    - \( \text{kill}[I] = \{ E \mid x \in E \} \)
  
  - if \( I \) is \( \text{addr } y \):
    - \( \text{gen}[I] = \{ \} \)
    - \( \text{kill}[I] = \{ E \mid x \in E \} \)
  
  - if \( I \) is \( \text{if } (x) \):
    - \( \text{gen}[I] = \{ \} \)
    - \( \text{kill}[I] = \{ \} \)
  
  - if \( I \) is \( \text{return } x \):
    - \( \text{gen}[I] = \{ \} \)
    - \( \text{kill}[I] = \{ \} \)
  
  - if \( I \) is \( x = f(y_1, \ldots, y_n) \):
    - \( \text{gen}[I] = \{ \} \)
    - \( \text{kill}[I] = \{ E \mid x \in E \} \)

- Transfer functions \( F_i(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)

- ... how about \( x = x \text{ OP } y \)?
Available expressions: Summary

- Lattice: \((2^E, \subseteq)\); has finite height
- Meet is set intersection, top element is \(E\)
- Is a forward dataflow analysis

Dataflow equations:
- \(\text{out}[B] = F_0(\text{in}[B]), \text{for all } B\)
- \(\text{in}[B] = \cap \{\text{out}[B'] | B' \in \text{pred}(B)\}, \text{for all } B\)
- \(\text{in}[B_0] = X_0\)

- Transfer functions: \(F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I]\)
  - are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution

Problem 3: Reaching Definitions

- Compute reaching definitions for each program point

  **Reaching definition** = definition of a variable whose assigned value
  may be observed at current program point in some execution of
  the program

- Dataflow information: sets of reaching definitions
- Example: definitions \(\{d2, d7\}\) may reach program point \(p\)
- Is a forward analysis

- Let \(D = \text{set of all definitions (assignments) in the program}\)
- Lattice \((D, \subseteq)\), where:
  - \(L = 2^D\) (power set of \(D\))
  - Partial order \(\subseteq\) is set inclusion: \(\supseteq\)
    \[ S_1 \subseteq S_2 \iff S_1 \supseteq S_2 \]
**RD: The Lattice**

- Consider set of expressions = \{d1, d2, d3\}
  where d1: x = y, d2: x=x+1, d3: z=y-x

- Partial order: \(\geq\)
- Set D is finite implies lattice has finite height
- Meet operator: \(\cup\) (set union)
- Top element: \(\emptyset\) (empty set)
- Smaller sets of reaching definitions = more precise analysis
- All definitions may reach current point = least precise

**RD: Dataflow equations**

- Equations:
  
  \[
  \begin{align*}
  \text{out}[B] &= F_B(\text{in}[B]), \text{ for all } B \\
  \text{in}[B] &= \cup\{\text{out}[B'] | B' \in \text{pred}(B)\}, \text{ for all } B \\
  \text{in}[B_s] &= X_0
  \end{align*}
  \]

- Meaning of intersection meet operator:
  
  “A definition reaches the entry of block B if it reaches the exit of at least one of its predecessor nodes”
RD: Transfer functions

- Define transfer functions for instructions
- General form of transfer functions:
  \[ F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]
  
  where:
  - \( \text{kill}[I] \) = definitions “killed” by \( I \)
  - \( \text{gen}[I] \) = definitions “generated” by \( I \)

- Meaning of transfer functions: “Reaching definitions after instruction \( I \) include: (1) reaching definitions before \( I \), but not killed by \( I \), and (2) reaching definitions generated by \( I \)”

RD: Transfer Functions

- Define \( \text{kill}/\text{gen} \) for each type of instruction
- If \( I \) is a definition \( d \) that defines \( x \):
  \[ \text{gen}[I] = \{d\} \quad \text{kill}[I] = \{d' \mid d' \text{ defines } x\} \]

  - If \( I \) is not a definition:
    \[ \text{gen}[I] = \{} \quad \text{kill}[I] = \{} \]

- Transfer functions \( F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)
- They are monotonic and distributive
  - For each \( F_I \), \( \text{kill}[I] \) and \( \text{gen}[I] \) are constants: they don’t depend on input information \( X \)
Reaching Definitions: Summary

- Lattice: \((2^\mathbb{D}, \supseteq)\); has finite height
- Meet is set union, top element is \(\emptyset\)
- Is a forward dataflow analysis
- Dataflow equations:
  \[
  \begin{align*}
  \text{out}[B] &= F_B(\text{in}[B]), \text{ for all } B \\
  \text{in}[B] &= \bigcup\{ \text{out}[B'] \mid B' \in \text{pred}(B) \}, \text{ for all } B \\
  \text{in}[B_0] &= X_0
  \end{align*}
  \]
- Transfer functions: \(F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I]\)
  - are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution

Implementation

- Lattices in these analyses = power sets
- Information in these analyses = subsets of a set
- How to implement subsets?
  1. Set implementation
     - Data structure with as many elements as the subset has
     - Usually list implementation
  2. Bitvectors:
     - Use a bit for each element in the overall set
     - Bit for element \(x\) is: 1 if \(x\) is in subset, 0 otherwise
     - Example: \(S = \{a, b, c\}\), use 3 bits
     - Subset \(\{a, c\}\) is 101, subset \(\{b\}\) is 010, etc.
Implementation Tradeoffs

- **Advantages of bitvectors:**
  - Efficient implementation of set union/intersection: set union is bitwise “or” of bitvectors set intersection is bitwise “and” of bitvectors
  - **Drawback:** inefficient for subsets with few elements

- **Advantage of list implementation:**
  - Efficient for sparse representation
  - **Drawback:** inefficient for set union or intersection

- In general, bitvectors work well if the size of the (original) set is linear in the program size

Problem 4: Constant Propagation

- Compute constant variables at each program point
- **Constant variable** = variable having a constant value on all program executions
- Dataflow information: sets of constant values
- Example: \{x=2, y=3\} at program point p
- Is a forward analysis

- Let \( V = \) set of all variables in the program, \( n_{\text{var}} = |V| \)
- Let \( N = \) set of integer numbers
- Use a lattice over the set \( V \times N \)
- Construct the lattice starting from a flat lattice for \( N \)
Flat lattice for $N$

- Lattice $L = (N \cup \{T, \perp\}, \sqsubseteq_F)$
  - $\perp \sqsubseteq_F n$, for all $n \in N$
  - Meaning of $T$ : “Not known to be constant”
  - $n \sqsubseteq_F T$, for all $n \in N$
  - Meaning of $\perp$ : “Known to be not constant”
- Distinct integer constants are not comparable

![Diagram of lattice](Diagram.png)

Note: meet of any two distinct numbers is $\perp$
Note: meet of any number and $T$ is that number

Constant folding lattice

- Flat lattice: $L = (N^*, \sqsubseteq_F)$, where $N^* = N \cup \{T, \perp\}$
- Constant folding lattice: $L' = (V \rightarrow N^*, \sqsubseteq_C)$
- Represent a function in $V \rightarrow N^*$ as a set of bindings:
  - $\{ v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \}$
- Define partial order $\sqsubseteq_C$ on $V \rightarrow N^*$ as:
  - $X \sqsubseteq_C Y$ iff $X(v) \sqsubseteq_F Y(v)$ for each variable $v$

$X = \{ v_1 = c_1, v_2 = c_2, \ldots \} \sqsubseteq_C$

$Y = \{ v_1 = c'_1, v_2 = c'_2, \ldots \}$

Constant folding example:
- $x = 230 \cdot 2 + 4$
- $x = 464$;
CF: Transfer Functions

• Transfer function for instruction $I$:
  \[ F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]
  where:
  \( \text{kill}[I] = \) constants “killed” by $I$
  \( \text{gen}[I] = \) constants “generated” by $I$

• If $I$ is $v = c$ (constant):
  - \( \text{gen}[I] = \{ v = c \} \)
  - \( \text{kill}[I] = \{ v = n \mid \text{for all } n \in \mathbb{N}^* \} \)

• If $I$ is $v = u + w$:
  - \( \text{gen}[I] = \{ v = k \} \)
  - \( \text{kill}[I] = \{ v = n \mid \text{for all } n \in \mathbb{N}^* \} \)
  - where
    \[ k = X(u) + X(w) \quad \text{if } X(u) \text{ and } X(w) \text{ are both constants} \]
    \[ k = D \quad \text{if } X(u) = D \text{ or } X(w) = D \quad k = \]
    \[ \top \quad \text{otherwise} \]

CF: Transfer functions

• Transfer function for instruction $I$:
  \[ F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]

• Here \( \text{gen}[I] \) is not constant, it depends on $X$

• However transfer functions are monotonic

• ... but are transfer functions distributive?
CF: Distributivity?

- Example:

\[
\begin{align*}
\{x=2, y=3, z=\top\} & \quad \text{\(x = 2\)} & \quad \{x=3, y=2, z=\top\} \\
\{x=2, y=3, z=\top\} & \quad \{x=2, y=3, z=\top\} & \quad \{x=3, y=2, z=\top\}
\end{align*}
\]

\[
\text{\(z = x+y\)} \quad \text{\(z = x+y\)}
\]

- At join point, apply meet operator
- Then use transfer function for \(z=x+y\)

- Dataflow result (MFP) at the end: \(\{x=\bot, y=\bot, z=\bot\}\)
- MOP solution at the end?
**CF: Distributivity?**

- Example:

\[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{z} \\
\hline
2 & 3 & 7 \\
3 & 2 & 5 \\
\end{array}
\]

\[z = x + y\]

\[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{z} \\
\hline
\bot & \bot & \bot \\
\bot & \bot & \bot \\
\end{array}
\]

- Dataflow result (MFP) at the end: \(\{x=\bot, y=\bot, z=\bot\}\)
- MOP solution at the end: \(\{x=\bot, y=\bot, z=5\}\)

**Reason for MOP \(\neq\) MFP:**

*transfer function \(F\) of \(z=x+y\) is not distributive!*

\[
F(X_1 \cap X_2) \neq F(X_1) \cap F(X_2)
\]

where \(X_1 = \{x=2, y=3, z=\top\}\) and \(X_2 = \{x=3, y=2, z=\top\}\)
Classification of analyses

- **Forward analyses**: information flows from
  - CFG entry block to CFG exit block
  - Input of each block to its output
  - Output of each block to input of its successor blocks
  - **Examples**: available expressions, reaching definitions, constant folding

- **Backward analyses**: information flows from
  - CFG exit block to entry block
  - Output of each block to its input
  - Input of each block to output of its predecessor blocks
  - **Example**: live variable analysis

Another classification

- **“may” analyses**:
  - information describes a property that **MAY** hold in **SOME** executions of the program
  - Usually: \( \Pi = U, \top = \emptyset \)
  - Hence, initialize info to empty sets
  - **Examples**: live variable analysis, reaching definitions

- **“must” analyses**:
  - information describes a property that **MUST** hold in **ALL** executions of the program
  - Usually: \( \Pi = \cap, \top = S \)
  - Hence, initialize info to the whole set
  - **Examples**: available expressions