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:
    \[ S_1 \subseteq S_2 \text{ if } S_1 \supseteq S_2 \]

LV: The Lattice
- Consider set of variables \( V = \{ x, y, z \} \)
- Partial order: \( \sqsubseteq \)
- Set \( V \) is finite implies lattice has finite height
- Meet operator: \( \sqcap \)
  (set union: \( \text{out}(B) \) is union of \( \text{in}(B') \), for all \( B' \in \text{succ}(B) \))
- Top element: \( \emptyset \)
  (empty set)
- Smaller sets of live variables = more precise analysis
- All variables may be live = least precise

LV: Dataflow Equations
- Equations:
  \[ \text{in}(B) = F_B(\text{out}(B)), \text{ for all } B \]
  \[ \text{out}(B) = \cup \{ \text{in}(B') | B' \in \text{succ}(B) \}, \text{ for all } B \]
  \[ \text{out}(B_0) = X_0 \]
- 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:
  \[ F_B(X) = (X - \text{def}[1]) \cup \text{use}[1] \]
  where:
  \( \text{def}[1] \) = set of variables defined (written) by \( I \)
  \( \text{use}[1] \) = set of variables used (read) by \( I \)
- Meaning of transfer functions:
  "Variables live before instruction \( I \) include: 1) variables live after \( I \), not written by \( I \), and 2) variables used by \( I \)"
**LV: Transfer Functions**

- Define def/use for each type of instruction
  - if \( x = y \) if OP \( z \): \( \text{use}[I] = \{y, z\} \) \( \text{def}[I] = \{x\} \)
  - if \( x = \text{OP} y \): \( \text{use}[I] = \{y\} \) \( \text{def}[I] = \{x\} \)
  - if \( x = y \) if \( z \): \( \text{use}[I] = \{z\} \) \( \text{def}[I] = \{x\} \)
  - if \( x = \text{addr} y \): \( \text{use}[I] = \{} \) \( \text{def}[I] = \{x\} \)
  - if \( \text{if} (x) \): \( \text{use}[I] = \{x\} \) \( \text{def}[I] = \{} \)
  - if \( \text{return} x \): \( \text{use}[I] = \{x\} \) \( \text{def}[I] = \{} \)
  - if \( x = f(y_1, \ldots, y_n) \): \( \text{use}[I] = \{y_1, \ldots, y_n\} \) \( \text{def}[I] = \{x\} \)
- Transfer functions \( F_x(X) = (X - \text{def}[I]) \cup \text{use}[I] \)
- For each \( F_x \), \( \text{def}[I] \) and \( \text{use}[I] \) are constants; they don't depend on input information \( X \)

**LV: Monotonicity**

- Are transfer functions \( F_x(X) = (X - \text{def}[I]) \cup \text{use}[I] \) monotonic?
- Because \( \text{def}[I] \) is constant, \( X - \text{def}[I] \) is monotonic:
  - \( X_1 \supset X_2 \) implies \( X_1 - \text{def}[I] \supset X_2 - \text{def}[I] \)
- Because \( \text{use}[I] \) is constant, \( Y \cup \text{use}[I] \) is monotonic:
  - \( Y_1 \supset Y_2 \) implies \( Y_1 \cup \text{use}[I] \supset Y_2 \cup \text{use}[I] \)
- Put pieces together: \( F_x(X) \) is monotonic
  - \( X_1 \supset X_2 \) implies
  - \( (X_1 - \text{def}[I]) \cup \text{use}[I] \supset (X_2 - \text{def}[I]) \cup \text{use}[I] \)

**LV: Distributivity**

- Are transfer functions \( F_x(X) = (X - \text{def}[I]) \cup \text{use}[I] \) distributive?
- Since \( \text{def}[I] \) 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 \( \text{use}[I] \) is constant: \( Y \cup \text{use}[I] \) is distributive:
  - \( (Y_1 \cup Y_2) \cup \text{use}[I] = (Y_1 \cup \text{use}[I]) \cup (Y_2 \cup \text{use}[I]) \)
  - Because: \( (a \cup b) \cup c = (a \cup c) \cup (b \cup c) \)
- Put pieces together: \( F_x(X) \) is distributive
  - \( F_x(X_1 \cup X_2) = F_x(X_1) \cup F_x(X_2) \)

**Live Variables: Summary**

- Lattice: \((2^E, \supseteq)\); has finite height
- Meet is set union, top is empty set
- Is a backward dataflow analysis
- Dataflow equations:
  - \( \text{in} (b) = F_x(\text{out}(b)) \), for all \( B \)
  - \( \text{out}(b) = \{e(b') | b' \in \text{succ}(b)\} \), for all \( B \)
  - \( \text{out}(b_0) = X_0 \)
- Transfer functions: \( F_x(X) = (X - \text{def}[I]) \cup \text{use}[I] \)
  - are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MIP 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 discussed earlier
- Dataflow information: sets of available expressions
- Example: expressions \( \{x+y, y-z\} \) are available at point \( p \)
- Is a forward analysis
- Let \( E \) = set of all expressions in the program
- Lattice \((L, \subseteq)\), where:
  - \( L = 2^E \) (power set of \( E \), i.e. set of all subsets of \( E \))
  - Partial order \( \subseteq \) is set inclusion
  - \( S_1 \subseteq S_2 \iff S_1 \subseteq S_2 \)

**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 variables = more precise analysis
- No available expressions = least precise
AE: Dataflow Equations

- Equations:
  \[ \text{out}[I] = F_b(\text{in}[I]), \text{ for all } B \]
  \[ \text{in}[B] = \bigcap \{ \text{out}[B'] | B' \preceq \text{pred}(B) \}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]

- 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:
  \[ F_0(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]

  where:
  \[ \text{kill}[I] = \text{expressions "killed" by } I \]
  \[ \text{gen}[I] = \text{new expressions "generated" by } I \]

- 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, not killed by I, and 2) expressions generated by I"

---

AE: Transfer Functions

- Define kill/gen for each type of instruction
  - if \( I = x \to y \): \( \text{gen}[I] = \{x \to y \} \)
  - if \( I = \text{OP} z : \text{gen}[I] = \{ \text{OP} z \} \)
  - if \( I = \text{addr y} : \text{gen}[I] = \{ y \} \)
  - if \( I = \text{if}(x) : \text{gen}[I] = \{ x \} \)
  - if \( I = \text{return} x : \text{gen}[I] = \{ x \} \)

- Transfer functions \( F_0(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)

- ... 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:
  \[ \text{out}[I] = F_b(\text{in}[I]), \text{ for all } B \]
  \[ \text{in}[B] = \bigcap \{ \text{out}[B'] | B' \preceq \text{pred}(B) \}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]

- Transfer functions: \( F_0(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)
  - 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 \((d_2, d_7)\) may reach program point \( p \)
- Is a forward analysis
- Let \( D = \) 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: \( \subseteq \)

---

RD: The Lattice

- Consider set of expressions = \((d_1, d_2, d_3)\)
  where \( d_1: x = y \), \( d_2: x = x + 1 \), \( d_3: z = y \cdot x \)

- Partial order: \( \subseteq \)
- 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:
  \[ \text{out}[I] = F_d(\text{in}[I]), \text{ for all } B \]
  \[ \text{in}[B] = \cup \{\text{out}(B') \mid B'\text{ pred}(B)\}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]

- 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 kill/gen for each type of instruction
  - If I is a definition d:
    \[ \text{gen}[I] = \{d\} \quad \text{kill}[I] = \{d' \mid d' \text{ defines } x\} \]
  - If I is not a definition:
    \[ \text{gen}[I] = \{\} \quad \text{kill}[I] = \{\} \]

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

RD: Transfer Functions

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

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

Reaching Definitions: Summary

- Lattice: \( (2^S, \supseteq) \); has finite height
- Meet is set union, top element is \( \emptyset \)
- Is a forward dataflow analysis
- Dataflow equations:
  \[ \text{out}(I) = F_d(\text{in}(I)), \text{ for all } B \]
  \[ \text{in}(B) = \cup \{\text{out}(B') \mid B'\text{ pred}(B)\}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]
- Transfer functions:
  \[ F_d(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \]
  - 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 Folding

- 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 \( p \)
- L = set of integer constants
- Use a lattice over the set \( L \times N \)
- Construct the lattice starting from a lattice for \( N \)
- Problem: \( (N, \leq) \) is not a complete lattice!
  ... Why?

Constant Folding Lattice

- Second try: lattice \( (N \cup \{T, \bot\}, \leq) \)
  - Where \( \bot \leq n \) for all \( n \in N \)
  - And \( n \leq T \) for all \( n \in N \)
  - Is complete!
- Problem:
  - Is incorrect for constant folding
  - Meet of two constants \( c \land d \) is \( \min(c,d) \)
  - Meet of different constants should be \( \bot \)
- Another problem: has infinite height ...
**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 (easy to prove)
- ... but are transfer functions distributive?

**CF: Distributivity**

- Example:
  \[
  \{x=2, y=3, z=\top\} \quad 
  \{x=3, y=2, z=\top\} \\
  \{x\top, y\top, z=\top\} \\
  \{x\top, y\top, z=\bot\} \\
  \{x=2, y=3, z=?\} \\
  \{x=3, y=2, z=?\} \\
  \{x=2, y=3, z=?\} \\
  \{x=3, y=2, z=?\}
  \]
- At join point, apply meet operator
- Then use transfer function for \( z=x+y \)

**CF: Distributivity**

- Example:
  \[
  \{x=2, y=3, z=\top\} \quad 
  \{x=3, y=2, z=\top\} \\
  \{x=\bot, y=\bot, z=\bot\} \\
  \{x=\bot, y=\bot, z=\top\} \\
  \{x=\bot, y=\bot, z=\bot\} \\
  \{x=2, y=3, z=?\} \\
  \{x=3, y=2, z=?\} \\
  \{x=\bot, y=\bot, z=?\}
  \]
- Dataflow result (MFP) at the end: \( \{x=\bot, y=\bot, z=\bot\} \)
- MOP solution at the end: \( \{x=\bot, y=\bot, z=5\} \)

**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
- **Examples**: live variable analysis

**Another Classification**

- "may" analyses:
  - Information describes a property that \textbf{MAY} hold in \textbf{SOME} executions of the program
  - Usually: \( \cap = \cup, \top = \emptyset \)
  - Hence, initialize info to empty sets
- **Examples**: live variable analysis, reaching definitions
- "must" analyses:
  - Information describes a property that \textbf{MUST} hold in \textbf{ALL} executions of the program
  - Usually: \( \cap = \cap, \top = \mathbb{N} \)
  - Hence, initialize info to the whole set
- **Examples**: available expressions