Dataflow Analysis

- 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:
  - FP ⊆ MFP ⊆ MOP ⊆ 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 \( x, y \) may be live at program point \( p \)
- Is a backward analysis
- Let \( V \) = set of all variables in the program
- Lattice \((L, \sqsubseteq)\), where:
  - \( L = 2^V \) (power set of \( V \), i.e. set of all subsets of \( V \))
  - Partial order \( \sqsubseteq \) is set inclusion: \( \sqsubseteq \)
  - \( S_1 \sqsubseteq S_2 \) if \( S_1 \subseteq S_2 \)
- Let \( V \) = set of all variables in the program
- Lattice \((L, \sqsubseteq)\), where:
  - \( L = 2^V \) (power set of \( V \), i.e. set of all subsets of \( V \))
  - Partial order \( \sqsubseteq \) is set inclusion: \( \sqsubseteq \)
  - \( S_1 \sqsubseteq S_2 \) if \( S_1 \subseteq S_2 \)

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:
  - \( \text{in}(B) = F_i(B), \) for all \( B \)
  - \( \text{out}(B) = \bigcup \{\text{in}(B) \mid B \in \text{succ}(B)\}, \) for all \( B \)
  - \( \text{out}(B_i) = X_0 \)
- 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, not written by I, and 2) variables used by I"

LV: Transfer Functions

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

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

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:
  \[ (Y_1 \cup Y_2) \cup \text{use}[I] = (Y_1 \cup \text{use}[I]) \cup (Y_2 \cup \text{use}[I]) \]
  because: \( (a \cup b) \cup c = (a \cup c) \cup (b \cup c) \)
- Put pieces together: \( F_I(X) \) is distributive:
  \[ F_I(X_1 \cup X_2) = F_I(X_1) \cup F_I(X_2) \]

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 \( E \) = set of all expressions in the program
- Lattice \( (E, \sqsubseteq) \), where:
  - \( L = 2^E \) (power set of \( E \), i.e. set of all subsets of \( E \))
  - Partial order \( \sqsubseteq \) is set inclusion: \( \sqsubseteq \)
  \[ S_1 \sqsubseteq S_2 \text{ iff } S_1 \subseteq S_2 \]

Live Variables: Summary

- Lattice: \( (2^S, \sqsubseteq) \); has finite height
- Meet is set union, top is empty set
- Is a backward dataflow analysis
- Dataflow equations:
  \[ \text{in}[B] = F_0(\text{out}[B]), \text{for all } B \]
  \[ \text{out}[B] = \{ \text{in}[B'] | B' \text{'s succ}(B) \}, \text{for all } B \]
- out[B] = \( S_0 \)
- 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
**AE: The Lattice**
- Consider set of expressions \( \{x^2, x+y, y-z\} \)
- Denote \( e = x^2 z, f = x+y, g = y-z \)
- 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:
  - \( \text{out}[I] = F_0(\text{in}[I]), \) for all \( B \)
  - \( \text{in}[B] = \cap \{\text{out}[B'] | B' \subset \text{pred}(B)\}, \) 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:
  - \( F_c(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)
  - 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 \( I \), not killed by \( I \), and 2) expressions generated by \( I \)"

**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 \( (D, \sqsubseteq) \), where:
  - \( L = 2^D \) (power set of \( D \))
  - Partial order \( \sqsubseteq \) = set inclusion: \( \sqsubseteq \)
  - \( S_1 \sqsubseteq S_2 \) if \( S_1 \subseteq S_2 \)
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-x
- Partial order: \emptyset
- Set D is finite implies lattice has finite height
- Meet operator: \bigcap (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}[1] = F_d(\text{in}[1]), \text{for all } B
  \text{in}[B] = \bigcup \{\text{out}[B'] | B' \subseteq \text{pred}(B)\}, \text{for all } B
  \text{in}[B_x] = 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 transfer functions for instructions
- General form of transfer functions:
  \text{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"

RD: Transfer Functions

- Define kill/gen for each type of instruction
- If I is a definition d:
  \text{gen}[I] = \{d\}
  \text{kill}[I] = \{d' | d' \text{ defines } x\}
- If I is not a definition:
  \text{gen}[I] = \{}
  \text{kill}[I] = \{}
- Transfer functions \text{F}_d(X) = (X - \text{kill}[I]) \cup \text{gen}[I]
- They are monotonic and distributive
  - For each \text{F}_d, \text{kill}[I] and \text{gen}[I] are constants: they don't depend on input information X

Reaching Definitions: Summary

- Lattice: (\mathbb{2}^5, \supseteq); has finite height
- Meet is set union, top element is \emptyset
- Is a forward dataflow analysis
- Dataflow equations:
  \text{out}[1] = F_d(\text{in}[1]), \text{for all } B
  \text{in}[B] = \bigcup \{\text{out}[B'] | B' \subseteq \text{pred}(B)\}, \text{for all } B
  \text{in}[B_x] = X_0
- Transfer functions: \text{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 \( p \)
- Is a forward analysis
  - Let \( V \) = set of all variables in the program, \( nvar = |V| \)
  - Let \( N \) = set of integer numbers
  - Use a lattice over the set \( V \times N \)
  - Construct the lattice starting from a lattice for \( N \)
- Problem: \( (N, \leq) \) is not a complete lattice!
  - ... why?

Constant Folding Lattice

- Second try: lattice \( (N \cup \{\top, \bot\}, \leq) \)
  - Where \( \bot \leq n \), for all \( n \in N \)
  - And \( n \leq \top \), for all \( n \in N \)
  - Is complete!
- Meaning:
  - \( v = \top \): don't know if \( v \) is constant
  - \( v = \bot \): \( v \) is not constant
- Problem:
  - Is incorrect for constant folding
  - Meet of two constants \( cd \) is \( \min(c,d) \)
  - Meet of different constants should be \( \bot \)
- Another problem: has infinite height ...

Constant Folding Lattice

- Solution: flat lattice \( L = (N \cup \{\top, \bot\}, \leq) \)
  - Where \( \bot \leq n \), for all \( n \in N \)
  - And \( n \leq \top \), for all \( n \in N \)
  - And distinct integer constants are not comparable
  - Note: meet of any two distinct numbers is \( \bot \! \)
**CF: Transfer Functions**

- Transfer function for instruction I:
  \[ F_I(x) = (x - \text{kill}[I]) \cup \text{gen}[I] \]
- where:
  \[ \text{kill}[I] = \text{constants "killed" by I} \]
  \[ \text{gen}[I] = \text{constants "generated" by I} \]
- \[ X[v] = \epsilon \iff \{v = \epsilon\} \in X \]
- If I is a constant:
  \[ \text{gen}[I] = \{v=\epsilon\} \]
  \[ \text{kill}[I] = \{v\} \times N^* \]
- If I is a 
  \[ \text{gen}[I] = \{v=\epsilon\} \]
  \[ \text{kill}[I] = \{v\} \times N^* \]
  where \( e = X[u] + X[w] \) if \( X[u] \) and \( X[w] \) are not \( \bot, \bot \)
  \( e = \bot, \text{if } X[u] = \bot \) or \( X[w] = \bot \)
  \( e = \top, \text{if } X[u] = \top \) or \( X[w] = \top \)

**CF: Distributivity**

- Example:
  \[ \{x=2, y=3, z=\top\} \rightarrow \{x=2, y=2, z=\top\} \]
  \[ z = x + y \]
  \[ \{x=2, y=2, z=\top\} \rightarrow \{x=2, y=2, z=\top\} \]
  \[ z = x + y \]

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

**CF: Distributivity**

- Example:
  \[ \{x=2, y=3, z=\top\} \rightarrow \{x=2, y=3, z=\top\} \]
  \[ z = x + y \]

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

**CF: Distributivity**

- Example:
  \[ \{x=2, y=3, z=\top\} \rightarrow \{x=2, y=3, z=\top\} \]
  \[ z = x + y \]

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

**Reason for MOP \neq MFP:**

- transfer function \( F \) of \( z=x+y \) is not distributive!
- \( F(X1 \cap X2) \neq F(X1) \cap F(X2) \)
- where \( X1 = \{x=2, y=3, z=\top\} \) and \( X2 = \{x=3, y=2, z=\top\} \)
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: \( \tau = \tau_1 \cup \tau_2 \)
  - 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: \( \tau = \tau_1 \cap \tau_2 \)
  - Hence, initialize info to the whole set
  - *Examples*: available expressions