Prelude

- What controversial list appeared this week in Atlantic Monthly?
  - 100 most influential Americans
- What profession is overwhelmingly represented?
  - Politicians – presidents, court justices
- What about scientists?
  - Thomas Edison (#9)
  - Eli Whitney (#27)
  - Albert Einstein (#32)
  - Jonas Salk (#34)
  - Robert Oppenheimer (#48)
  - James Watson (#68)
- What about computer scientists?
  - Does Bill Gates (#54) count?
    - Number 1?
    - Abraham Lincoln
    - (Then Washington and Jefferson)
    - Who’s missing?
    - J F K

Dataflow analysis

- Dataflow analysis
  - A common framework for such analysis
  - Computes information at each program point
  - Conservative: characterizes all possible program behaviors
- Methodology
  - Describe the information (e.g., live variable sets) using a structure called a lattice
  - Build a system of equations based on:
    - How each statement affects information
    - How information flows between basic blocks

Dataflow Analysis

- Dataflow analysis
  - Solving the system of equations
  - Iteratively computes maximal fixed point (MFP)
- Terminates because transfer functions are monotonic and lattice has finite height
- Other possible solutions: FP, MOP, IDEAL
  - FP is any fixed point
  - MOP is meet-over-all-paths
  - IDEAL is “perfect” information

Kinds of solutions

- All are safe solutions, but some are more precise:
  \[
  \text{FP} \subseteq \text{MFP} \subseteq \text{MOP} \subseteq \text{IDEAL}
  \]
- Bad news: MOP and IDEAL are intractable
  - But, MFP = MOP if transfer functions are distributive
- 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 \( \subseteq \)
- \( S_1 \subseteq S_2 \) iff \( S_1 \supseteq S_2 \)

LV: The Lattice

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

LV: Dataflow Equations

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

- Define transfer functions for instructions
- General form of transfer functions:
  \[
  F_I(X) = (X - \text{def}[I]) \cup \text{use}[I]
  \]
  where:
  - \( \text{def}[I] = \) set of variables defined (written) by \( I \)
  - \( \text{use}[I] = \) 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
- General form of transfer functions:
  \[
  F_I(X) = (X - \text{def}[I]) \cup \text{use}[I]
  \]
  where:
  - \( \text{def}[I] = \) set of variables defined (written) by \( I \)
  - \( \text{use}[I] = \) 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 \( I \)"
- 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: Transfer Functions

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

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

- For each \( F_I \), \( \text{def}[I] \) and \( \text{use}[I] \) 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 \( \text{def}[I] \) 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 \( \text{use}[I] \) 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
  - \( X_1 \supseteq X_2 \) 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 \( \text{def}[I] \) 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: \( F_I(X) \) 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:
    \( \text{in}[B] = F_B(\text{out}[B]), \text{for all } B \)
    \( \text{out}[B] = \bigcup \{ \text{in}[B'] | B' \in \text{succ}(B) \}, \text{for all } B \)
  - 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

- Available expression = a previously evaluated expression that would have the same value if re-evaluated at the current point
- Dataflow information: sets of available expressions
- Example: \( \text{exprs} \{x+y, y-z\} \) are available at point \( p \)
- Let \( E \) = set of all available 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: \( \subseteq \)
  - \( S_1 \subseteq S_2 \iff S_1 \subseteq S_2 \)

AE: The Lattice

- Consider set of expressions = \{x*z, x+y, y-z\}
- Denote \( e = x*z, f = x+y, g = y-z \)
- Larger sets of available variables = more precise analysis
- No available expressions = least precise
  - 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)
AE: Dataflow Equations

- Equations:
  \[ \text{out}[I] = \mathcal{F}_B(\text{in}[I]), \text{ for all } B \]
  \[ \text{in}[B] = \cap \{ \text{out}[B'] | B' \in \text{pred}(B) \}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]

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

- 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"

Available Expressions

- Lattice: \((2E, \subseteq)\); has finite height
- Is a forward dataflow analysis
- Dataflow equations:
  \[ \text{out}[I] = \mathcal{F}_B(\text{in}[I]), \text{ for all } B \]
  \[ \text{in}[B] = \cap \{ \text{out}[B'] | B' \in \text{pred}(B) \}, \text{ for all } B \]
  \[ \text{in}[B_0] = X_0 \]

- 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
- \text{Reaching definition} = \text{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 \(D = \text{set of all definitions (assignments) in the program}\)

- Lattice \((D, \subseteq)\), where:
  - \(L = 2^D\) (power set of D)
  - Partial order \(\subseteq\) is set inclusion: \(\subseteq\)

- \(S_1 \subseteq S_2 \iff S_1 \supseteq S_2\)
RD: The Lattice
- Consider set of expressions = \{d1, d2, d3\}
  where d1: x = y, d2: x=x+1, d3: z=y-x
- Smaller sets of reaching definitions = more precise analysis
- All definitions may reach current point = least precise
- Partial order: \( \supseteq \)
- Set D is finite implies lattice has finite height
- Meet operator: \( \cup \) (set union)
- Top element: \( \emptyset \) (empty set)

RD: Dataflow Equations
- Equations:
  \[
  \text{out}[B] = F_B(\text{in}[B]), \text{for all } B
  \]
  \[
  \text{in}[B] = \cup \{ \text{out}[B'] | B' \in \text{pred}(B) \}, \text{for all } B
  \]
  \[
  \text{in}[B_0] = X_0
  \]
- 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 transfer functions for instructions
- General form of transfer functions:
  \[
  F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I]
  \]
  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:
  \[
  \text{gen}[I] = \{d\}
  \]
  \[
  \text{kill}[I] = \{d' | d' defines x\}
  \]
- If I is not a definition:
  \[
  \text{gen}[I] = \emptyset
  \]
  \[
  \text{kill}[I] = \emptyset
  \]
- 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
- Lattice: \((2^D, \supseteq)\); has finite height
- Meet is set union, top element is \( \emptyset \)
- Is a forward dataflow analysis
- Dataflow equations:
  \[
  \text{out}[B] = F_B(\text{in}[B]), \text{for all } B
  \]
  \[
  \text{in}[B] = \cup \{ \text{out}[B'] | B' \in \text{pred}(B) \}, \text{for all } B
  \]
  \[
  \text{in}[B_0] = X_0
  \]
- 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

Implementation
- Lattices in these analyses = power sets
- Information in these analyses = subsets of a set
- How to implement subsets?
- Set implementation
  - Data structure with as many elements as the subset has
  - Usually list implementation
- 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

- **Pros and cons 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
- **Pros and cons 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

Next time…

- Only three more lectures!
- **Topics:**
  - Memory management
  - Linking and loading
  - Compiling functional languages
  - More optimizations (loops)