Lecture 28: Dataflow Analysis Instances
2 Apr 08
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:
  \[ \text{FP} \subseteq \text{MFP} \subseteq \text{MOP} \subseteq \text{IDEAL} \]
• MFP = MOP if transfer functions are distributive
• 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 \((L, \sqsubseteq)\), where:
  - \( L = 2^V \) (power set of \( V \), i.e., set of all subsets of \( V \))
  - Partial order \( \sqsubseteq \) is set inclusion: \( \sqsupseteq \)
    \[ S_1 \subseteq S_2 \iff S_1 \supseteq S_2 \]
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: \( \sqcap \)
  (set union: out\([B]\) is union of 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:
  \[
  \text{in}[B] = F_B(\text{out}[B]), \text{ for all } B
  \]
  \[
  \text{out}[B] = \bigcup \{\text{in}[B'] | B' \in \text{succ}(B)\}, \text{ for all } B
  \]
  \[
  \text{out}[B_e] = X_0
  \]

• 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:
  - \( \text{def}[I] \) = 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 \( I \), 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 \( x = y \text{ OP } z \): \quad \text{use}[I] = \{y, z\} \quad \text{def}[I] = \{x\}
if I is \( x = \text{ OP } y \) : \quad \text{use}[I] = \{y\} \quad \text{def}[I] = \{x\}
if I is \( x = y \) : \quad \text{use}[I] = \{y\} \quad \text{def}[I] = \{x\}
if I is \( x = \text{addr } y \) : \quad \text{use}[I] = \{} \quad \text{def}[I] = \{x\}
if I is if (x) : \quad \text{use}[I] = \{x\} \quad \text{def}[I] = \{}
if I is return x : \quad \text{use}[I] = \{x\} \quad \text{def}[I] = \{}
if I is \( x = f(y_1, \ldots, y_n) \) : \quad \text{use}[I] = \{y_1, \ldots, y_n\} \quad \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: Distributivity

• Are transfer functions: \( F_1(X) = (X - \text{def}[I]) \cup \text{use}[I] \) 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])
  
  \text{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 \cap \text{use}[I])
  
  \text{because: } (a \cup b) \cup c = (a \cup c) \cup (b \cup c)
  \]

• Put pieces together: \( F_1(X) \) is distributive
  \[
  F_1(X_1 \cup X_2) = F_1(X_1) \cup F_1(X_2)
  \]
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:
  \[
  \text{in}[B] = F_B(\text{out}[B]), \text{ for all } B \\
  \text{out}[B] = \bigcup \{ \text{in}[B'] | B' \in \text{succ}(B) \}, \text{ for all } B \\
  \text{out}[B_e] = X_0
  \]

• 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 \( E \) = 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: \( \sqsupseteq \)
    \[ S_1 \sqsubseteq S_2 \text{ iff } S_1 \supseteq S_2 \]
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:
  \[ \text{out}[B] = F_B(\text{in}[B]), \text{ for all } B \]
  \[ \text{in}[B] = \cap \{\text{out}[B'] \mid B' \in \text{pred}(B)\}, \text{ for all } B \]
  \[ \text{in}[B_s] = X_0 \]

• 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 \( I \), but not killed by \( I \), and (2) expressions generated by \( I \)”
AE: Transfer Functions

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

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

- \( \ldots \text{ how about } x = x \, \text{OP} \, y? \)
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:
  \[
  \text{out}[B] = F_B(\text{in}[B]), \text{ for all } B
  \]
  \[
  \text{in}[B] = \cap \{\text{out}[B'] | B' \in \text{pred}(B)\}, \text{ for all } B
  \]
  \[
  \text{in}[B_s] = X_0
  \]

- 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 \(D\) = 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: \supseteq

- Set D is finite implies lattice has finite height

- Meet operator: \bigvee
  (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:

\[ \text{out}[B] = F_B(\text{in}[B]), \text{ for all } B \]
\[ \text{in}[B] = \bigcup \{ \text{out}[B'] \mid B' \in \text{pred}(B) \}, \text{ for all } B \]
\[ \text{in}[B_s] = X_0 \]

• 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, but 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 \( d \) 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 \textit{constants}: they don't depend on input information \( X \)
Reaching Definitions: Summary

- Lattice: \((2^P, \supseteq)\); has finite height
- Meet is set union, top element is \(\emptyset\)
- Is a forward dataflow analysis
- Dataflow equations:
  \[
  \text{out}[B] = F_B(\text{in}[B]), \text{ for all } B
  \]
  \[
  \text{in}[B] = \cup \{\text{out}[B'] | B' \in \text{pred}(B)\}, \text{ for all } B
  \]
  \[
  \text{in}[B_s] = X_0
  \]
- 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?

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

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

\[
\begin{array}{c}
\top \\
| \\
-2 \quad -1 \quad 0 \quad 1 \quad 2 \quad \ldots
\end{array}
\]

Note: meet of any two distinct numbers is \( \bot \)

Note: meet of any number and \( \top \) is that number
Constant Folding Lattice

- **Flat lattice:** $L = (N^*, \sqsubseteq_F)$, where $N^* = N \cup \{\top, \bot\}$

- **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 $v$

\[
X = \{ v_1 = c_1, v_2 = c_2, \ldots \} \sqsubseteq_C \\
\quad \sqsubseteq_F \quad \sqsubseteq_F \\
Y = \{ v_1 = c'_1, v_2 = c'_2, \ldots \}
\]
CF: Transfer Functions

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

  where:

  $\text{kill}[I] = \text{constants “killed” by } I$
  
  $\text{gen}[I] = \text{constants “generated” by } I$

• 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

  $k = X(u)+X(w)$ if $X(u)$ and $X(w)$ are both constants

  $k = \top$ if $X(u) = \bot$ or $X(w) = \bot$

  $k = \top$ 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:

\[
\begin{align*}
\{x=2, \ y=3, \ z=\top\} & \quad \text{At join point, apply meet operator} \\
\{x=3, \ y=2, \ z=\top\} & \\
\{x=\?, \ y=\?, \ z=?\} & \\
\{x=\?, \ y=\?, \ z=?\} & \\
\end{align*}
\]

- Then use transfer function for \(z=x+y\)
CF: Distributivity?

• Example:

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

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

• Example:

{x=2, y=3, z=⊤} - {x=2, y=3, z=⊥} - {x=3, y=2, z=⊤} - {x=⊥, y=⊥, z=⊤}

• Dataflow result (MFP) at the end: {x=⊥, y=⊥, z=⊥}

• MOP solution at the end: {x=⊥, y=⊥, z=5}!
**CF: Distributivity?**

- **Example:**

  \[
  \begin{align*}
  x &= 2 \\
y &= 3 \\
z &= x + y
  \end{align*}
  \]

  \[
  \begin{align*}
x &= 3 \\
y &= 2 \\
z &= x + y
  \end{align*}
  \]

- **Reason for MOP ≠ 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: \( \Pi = \cup, \top = \emptyset \)
  - 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: \( \Pi = \cap, \top = \mathbb{S} \)
  - Hence, initialize info to the whole set
  - Examples: available expressions