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

LV: The Lattice

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

LV: Dataflow Equations

- General dataflow equations \( (X_0 \text{ is information at the end of exit basic block}) \):
  - \( \text{in}(B) = F_B(\text{out}(B)) \), for all \( B \)
  - \( \text{out}(B) = \bigcap \{ \text{in}(B') | B' \in \text{succ}(B) \} \), for all \( B \)
  - \( \text{out}(B_0) = X_0 \)
- Replace meet with set union:
  - \( \text{in}(B) = F_B(\text{out}(B)) \), for all \( B \)
  - \( \text{out}(B) = \bigcup \{ \text{in}(B') | B' \in \text{succ}(B) \} \), 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:
  - \( f_B(X) = (X - \text{def}(I)) \cup \text{use}(I) \)
    - 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 \( I \): 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 Is = y OP z:
  - use[I] = (y, z)
  - def[I] = (x)
- If Is = y:
  - use[I] = (y)
  - def[I] = (x)
- If Is = addr y:
  - use[I] = ()
  - def[I] = (x)
- If Is = f(y → y0):
  - use[I] = (y0 → y)
  - def[I] = (x)
- Transfer functions F_x = (X - def[I]) ∪ use[I]
- For each F_x, def[I] and use[I] are constants: they don't depend on input information X.

LV: Monotonicity

- Are transfer functions F_x = (X - def[I]) ∪ use[I] monotonic?
- Because def[I] is constant, X - def[I] is monotonic:
  - X1 ⊆ X2 implies X1 - def[I] ⊆ X2 - def[I]
- Because use[I] is constant, Y ∪ use[I] is monotonic:
  - Y1 ⊆ Y2 implies Y1 ∪ use[I] ⊆ Y2 ∪ use[I]
- Put pieces together: F_x = (X - def[I]) ∪ use[I]
  - are monotonic and distributive:
  - X1 ⊆ X2 implies
    (X1 - def[I]) ∪ use[I] ⊆ (X2 - def[I]) ∪ use[I]

LV: Distributivity

- Are transfer functions F_x = (X - def[I]) ∪ use[I] distributive?
- Since def[I] is constant, X - def[I] is distributive:
  - (X1 ∪ X2) - def[I] = (X1 - def[I]) ∪ (X2 - def[I])
  - because: (a ∪ b) - c = (a - c) ∪ (b - c)
- Since use[I] is constant, Y ∪ use[I] is distributive:
  - (Y1 ∪ Y2) ∪ use[I] = (Y1 ∪ use[I]) ∪ (Y2 ∪ use[I])
  - because: (a ∪ b) ∪ c = (a ∪ c) ∪ (b ∪ c)
- Put pieces together: F_x = (X - def[I]) ∪ use[I]
  - are monotonic and distributive:

Live Variables: Summary

- Lattice: (X', ⊆)
- has finite height
- Meet is union, top is empty set
- Is a backward dataflow analysis
- Dataflow equations:
  - F[B] = F[Out(B)], for all B
  - Out[B] = ∪ {h[B] | B ∈ sub(B)), for all B
  - Out[B0] = X0
- Transfer functions F_x = (X - def[I]) ∪ 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 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 (E, ⊆), where:
  - L = 2^|E| (power set of E; i.e. set of all subsets of E)
  - Partial order ⊆ is set inclusion
  - S ⊆ S' if S ⊆ S'
- Top element: (e,f,g)
-Meet operator: ∩
  - set intersection
- Top element: (e,f,g)
  - (set of all expressions)
- Alternate sets of available variables = more precise analysis
- No available expressions = least precise

AE: The Lattice

- Consider set of expressions = {x+y, x+y, y-z}
- Denote e = x+y, f = x+y, g = y-z
- Partial order ⊆
- Top element: (e,f,g)
  - set of all expressions
- Alternate sets of available variables = more precise analysis
- No available expressions = least precise
AE: Dataflow Equations

- General forward dataflow equations ($X_0$ is information at beginning of entry basic block):
  \[ \text{out}[I] = F_0(\text{in}[I]), \text{for all B} \]
  \[ h[B] = \bigcap \{ \text{out}[E] \mid B \subseteq \text{pred}(E) \}, \text{for all B} \]
  \[ h[B_0] = X_0 \]
- Replace meet with set intersection:
  \[ \text{out}[I] = F_0(\text{in}[I]), \text{for all B} \]
  \[ h[B] = \bigcap \{ \text{out}[E] \mid B \subseteq \text{pred}(E) \}, \text{for all B} \]
  \[ h[B_0] = X_0 \]
- 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 kill and gen for each type of instruction
  \[ \text{if} x = y \text{ op } z : \text{gen}[I] = \{ y \text{ op } z \} \]
  \[ \text{kill}[I] = \{ E \mid x \in E \} \]
  \[ \text{if} x = \text{op } y : \text{gen}[I] = \{ \text{op } y \} \]
  \[ \text{kill}[I] = \{ E \mid x \in E \} \]
  \[ \text{if} x = \text{addr } y : \text{gen}[I] = \{ y \} \]
  \[ \text{kill}[I] = \{ E \mid x \in E \} \]
  \[ \text{if} \text{if } (x) : \text{gen}[I] = \{ \} \]
  \[ \text{kill}[I] = \{ \} \]
  \[ \text{return } x : \text{gen}[I] = \{ x \} \]
  \[ \text{kill}[I] = \{ \} \]
  \[ \text{if } \text{if } \text{f}(y_1, \ldots, y_n) : \text{gen}[I] = \{ \} \]
  \[ \text{kill}[I] = \{ E \mid x \in E \} \]
- Transfer functions $F_X = (X - \text{kill}[I]) \cup \text{gen}[I]$
- For each $F$, kill and gen are constants: they don't depend on input information $X$

AE: Monotonicity

- Are transfer functions $F_X = (X - \text{kill}[I]) \cup \text{gen}[I]$ monotonic?
- Because $\text{kill}[I]$ is constant, $X - \text{kill}[I]$ is monotonic:
  $X_1 \subseteq X_2$ implies $X_1 - \text{kill}[I] \subseteq X_2 - \text{kill}[I]$
- Because gen[I] is constant, $Y \cup \text{gen}[I]$ is monotonic:
  $Y_1 \subseteq Y_2$ implies $Y_1 \cup \text{gen}[I] \subseteq Y_2 \cup \text{gen}[I]$
- Put pieces together: $F_X$ is monotonic
  $X_1 \subseteq X_2$ implies
  $(X_1 - \text{kill}[I]) \cup \text{gen}[I] \subseteq (X_2 - \text{kill}[I]) \cup \text{gen}[I]$

AE: Distributivity

- Are transfer functions $F_X = (X - \text{kill}[I]) \cup \text{gen}[I]$ distributive?
- Since $\text{kill}[I]$ is constant, $X - \text{kill}[I]$ is distributive:
  $X_1 \cap X_2 - \text{def}[I] = (X_1 - \text{def}[I]) \cap (X_2 - \text{def}[I])$ because: $(a \cap b) - c = (a - c) \cap (b - c)$
- Since gen[I] is constant, $Y \cup \text{gen}[I]$ is distributive:
  $(Y_1 \cap Y_2) \cup \text{gen}[I] = (Y_1 \cup \text{gen}[I]) \cap (Y_2 \cup \text{gen}[I])$ because: $(a \cap b) + c = (a + c) \cap (b + c)$
- Put pieces together: $F_X$ is distributive
  $F_X(X_1 \cap X_2) = F_X(X_1) \cap F_X(X_2)$

Available Expressions: Summary

- Lattice: $(S, \subseteq)$ has finite height
- Meet is set intersection, top element is $E$
- Is a forward dataflow analysis
- Dataflow equations:
  \[ \text{out}[I] = F_0(\text{in}[I]), \text{for all B} \]
  \[ h[B] = \bigcap \{ \text{out}[E] \mid B \subseteq \text{pred}(E) \}, \text{for all B} \]
  \[ h[B_0] = X_0 \]
- Transfer functions $F_X = (X - \text{kill}[I]) \cup \text{gen}[I]$
  - Are monotonic and distributive
- Iterative solving of dataflow equation:
  - Terminates
  - Computes MOP solution

CS 4124/13 Spring 2002 Introduction to Compilers
### 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 \((D, \leq)\), where:
  - \(L = 2^D\) (power set of \(D\))
  - Partial order \(\subseteq\) is set inclusion: \(A \subseteq B\)
  - \(S_1 \subseteq S_2 \iff S_1 \supseteq S_2\)

### RD: The Lattice
- Consider set of expressions \(d\) = \(d_1, d_2, d_3\)
  - where \(d_1: x = y, d_2: x = x + 1, d_3: z = y - x\)
  - Partial order: \(\subseteq\)
  - Set \(D\) is finite implies lattice has finite height
  - Meet operator: \(\cap\) (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
- General forward dataflow equations \((X, i)\) is information at beginning of entry basic block:
  - \(\text{out}[i] = F_0(i[i]), \text{for all } B\)
  - \(h[B] = \bigcap \{\text{out}[B] | B \in \text{pred}(B)\}, \text{for all } B\)
  - \(h[B] = X_i\)
- Replace meet with set union:
  - \(\text{out}[i] = F_0(h[i]), \text{for all } B\)
  - \(h[B] = \bigcup \{\text{out}[B] | B \in \text{pred}(B)\}, \text{for all } B\)
  - \(h[B] = X_i\)
- 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:
    \(F_i(X) = (X - \text{killed}_i) \cup \text{generated}_i\)
    - \(\text{killed}_i = \text{definitions "killed" by } i\)
    - \(\text{generated}_i = \text{definitions "generated" by } i\)
- 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 \(\text{killed}_i\) for each type of instruction
  - If \(i\) is a definition \(d\):
    \(\text{gen}_i = \{d\} \iff d\) defines \(x\)
  - If \(i\) is not a definition:
    \(\text{gen}_i = \{\}\)
- \(\text{killed}_i = \{\}\)
- Transfer functions \(F_i(X) = (X - \text{killed}_i) \cup \text{gen}_i\)
  - For each \(F_i\), \(\text{killed}_i\) and \(\text{gen}_i\) are constants: they don’t depend on input information \(X\)

### RD: Monotonicity
- Transfer function: \(F_i(X) = (X - \text{killed}_i) \cup \text{gen}_i\)
  - \(F_i(X)\) is monotonic:
    \(X_1 \supseteq X_2\) implies \((X_1 - \text{killed}_i) \cup \text{gen}_i \supseteq (X_2 - \text{killed}_i) \cup \text{gen}_i\)
  - \(F_i(X)\) is distributive:
    \(F_i(X_1 \cup X_2) = F_i(X_1) \cup F_i(X_2)\)
  - Same reasoning as before
Reaching Definitions: Summary

- Lattice: \( (2^\omega, \subseteq) \); has finite height
- Meet is set union, top element is \( \varnothing \)
- Is a forward dataflow analysis
- Dataflow equations:
  \[
  \text{out}([I]) = f_B ([I]), \text{for all } B \\
  h([B]) = \bigcup \{ \text{out}([B]) \mid B \in \text{pred}([B]) \}, \text{for all } B \\
  h([B]) = X_0
  \]
- Transfer functions \( f_B(X) = (X - \text{kill}[I]) \cup \text{gen}[I] \)
- 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. Bivectors:
     - 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 bivectors:
  - Efficient implementation of set union/intersection:
    - Set union is bitwise "or" of bivectors
    - Set intersection is bitwise "and" of bivectors
  - Drawbacks: inefficient for subsets with few elements
- Advantage of list implementation:
  - Efficient for sparse representation
  - Drawbacks: inefficient for set union or intersection
- In general, bivectors 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 \( V = \text{set of all variables in the program, } \text{var} = \{V\} \)
  - Let \( N = \text{set of integer constants} \)
  - 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
    - ... because there is no LUB(\( \varnothing \)) and GLB(\( N \))

Constant Folding Lattice

- Second try: lattice \( (\mathbb{N}_U \{ \top, \bot \}, \leq) \)
  - \( \top \)
  - \( \bot \)
  - \( \leq \)
  - \( \text{Is complete?} \)
  - Meaning:
    - \( v = \top \): don't know \( f \) is constant
    - \( v = \bot \): \( v \) is not constant
- Another problem: has infinite height ...

Constant Folding Lattice

- Second try: lattice \( (\mathbb{N}_U \{ \top, \bot \}, \leq) \)
  - \( \top \)
  - \( \bot \)
  - \( \leq \)
  - \( \text{Is complete?} \)
- Problem:
  - Is incorrect for constant folding
  - Meet of two constants \( c,d \) is \( \text{min}(c,d) \)
  - Meet of different constants should be \( \bot \)
- Another problem: has infinite height ...

CS 412413 Spring 2002 Introduction to Compilers
Constant Folding Lattice

- Solution: flat lattice $L = (N \cup \{T, \bot\}, \preceq)$
  - Where $\bot \preceq n$, for all $n \in N$
  - And $n \preceq T$, 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:
  $\text{kill}[I] = \text{constants "killed" by I}$
  $\text{gen}[I] = \text{constants "generated" by I}$
  $X[v] = C \in N^*$ if $(v = C) \in X$
  If I is $\text{v} = C$ (constant):
    $\text{gen}[I] = \langle v = C \rangle$
  $\text{kill}[I] = \langle v \rangle \times N^*$

- If I is $v = u + w$:
  $\text{gen}[I] = \langle v = u \rangle$
  $\text{kill}[I] = \langle v \rangle \times N^*$

where $u = X[u] + X[w]$, if $X[u]$ and $X[w]$ are not $T$, $\bot$
  - $e = \bot$, if $X[u] = \bot$ or $X[w] = \bot$
  - $e = T$, if $X[u] = T$ and $X[w] = T$

CF: Distributivity

- Example:
  $\{x = 2, y = 3, z = T\} \vdash x = 2$  
  $\{x = 3, y = 2, z = T\} \vdash x = 2, y = 3$

- At join point, apply meet operator:
  - Then use transfer function for $x = x + y$

CF: Distributivity

- Example:
  $\{x = 2, y = 3, z = T\} \vdash x = 2$  
  $\{x = 3, y = 2, z = T\} \vdash x = 2, y = 3$

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

- MOP solution at the end:
  $\{x = \bot, y = \bot, z = 5\}$
CF: Distributivity

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

- 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: \( \Gamma = \emptyset \)
  - 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: \( \Gamma = \emptyset \)
  - Hence, initialize info to the whole set
  - Examples: available expressions