COMP 181

Lecture 22

More on analysis and optimization

November 28, 2006

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?
- JFK

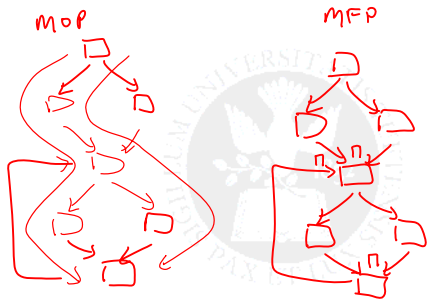
Dataflow analysis

- 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



Kinds of solutions

- All are safe solutions, but some are more precise:

  FP ⊑ MFP ⊑ MOP ⊑ 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

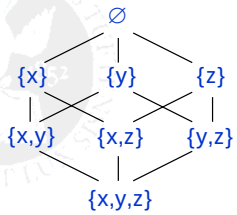
Problem 1: Live Variables

- 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, ⊑), where:
  - $L = 2^V$ (power set of V, i.e. set of all subsets of V)
  - Partial order ⊑ is set inclusion: ⊇

    $S_1 \sqsubseteq S_2$ 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: ⊇
- Set V is finite implies lattice has finite height
- Meet operator: ∪ (set union: out[B] is union of in[B'], for all B'∈succ(B)
- Top element: ∅ (empty set)



LV: Dataflow Equations

- Equations:

  in[B] = F_B(out[B]), for all B

  out[B] = ∪ {in[B'] | B'∈succ(B)}, for all B

  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:

  $F_I(X) = (X – def[I]) \cup use[I]$

  where:

  def[I] = set of variables defined (written) by I

  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

if I is x = y OP z :	use[I] =	def[I] =
if I is x = OP y :	use[I] =	def[I] =
if I is x = y :	use[I] =	def[I] =
if I is x = addr y :	use[I] =	def[I] =
if I is if (x) :	use[I] =	def[I] =
if I is return x :	use[I] =	def[I] =
if I is x = f(y_1,…, y_n) :	use[I] =	def[I] =

- Transfer functions $F_I(X) = (X – def[I]) \cup use[I]$
- 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 OP z :	use[I] = {y, z}	def[I] = {x}
if I is x = OP y :	use[I] = {y}	def[I] = {x}
if I is x = y :	use[I] = {y}	def[I] = {x}
if I is x = addr y :	use[I] = {}	def[I] = {x}
if I is if (x) :	use[I] = {x}	def[I] = {}
if I is return x :	use[I] = {x}	def[I] = {}
if I is x = f(y_1,…, y_n) :	use[I] = {y_1,…, y_n}	def[I] = {x}

- Transfer functions $F_I(X) = (X – def[I]) \cup use[I]$
- For each F_I, def[I] and use[I] are constants: they don't depend on input information X

LV: Monotonicity

- Are transfer functions: $F_I(X) = (X – def[I]) \cup use[I]$ monotonic?
- Because def[I] is constant, X – def[I] is monotonic:
  $X1 \supseteq X2$ implies $X1 – def[I] \supseteq X2 – def[I]$
- Because use[I] is constant, $Y \cup use[I]$ is monotonic:
  $Y1 \supseteq Y2$ implies $Y1 \cup use[I] \supseteq Y2 \cup use[I]$
- Put pieces together: $F_I(X)$ is monotonic
  $X1 \supseteq X2$ implies
  $(X1 – def[I]) \cup use[I] \supseteq (X2 – def[I]) \cup use[I]$

LV: Distributivity

- Are transfer functions: $F_I(X) = (X – def[I]) \cup use[I]$ distributive?
- Since def[I] is constant: X – def[I] is distributive:
  $(X1 \cup X2) – def[I] = (X1 – def[I]) \cup (X2 – def[I])$
  because: $(a \cup b) – c = (a – c) \cup (b – c)$
- Since use[I] is constant: $Y \cup use[I]$ is distributive:
  $(Y1 \cup Y2) \cup use[I] = (Y1 \cup use[I]) \cup (Y2 \cup use[I])$
  because: $(a \cup b) \cup c = (a \cup c) \cup (b \cup c)$
- Put pieces together: $F_I(X)$ is distributive
  $F_I(X1 \cup X2) = F_I(X1) \cup F_I(X2)$

Live Variables: Summary

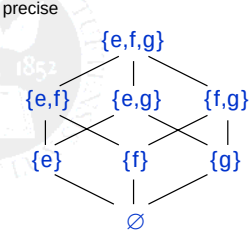
- Lattice: $(2^V, \supseteq)$; has finite height
- Meet is set union, top is empty set
- Is a backward dataflow analysis
- Dataflow equations:
  in[B] = F_B(out[B]), for all B
  out[B] = $\cup$ {in[B'] | B'$\in$succ(B)}, for all B
  out[B_e] = X_0
- Transfer functions: $F_I(X) = (X – def[I]) \cup use[I]$
  - 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: exprs {x+y, y-z} are available at point p
- 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: $\subseteq$
    $S_1 \sqsubseteq S_2$ 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:

  $out[I] = F_B(in[I])$, for all B

  $in[B] = \cap \{out[B'] \mid B' \in pred(B)\}$, for all B

  $in[B_s] = X_0$

- 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:

  $F_I(X) = (X - kill[I]) \cup gen[I]$

  where:

  kill[I] = expressions "killed" by I

  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"

AE: Transfer Functions

- Define kill/gen for each type of instruction

if I is x = y OP z :	gen[I] =	kill[I] =
if I is x = OP y :	gen[I] =	kill[I] =
if I is x = y :	gen[I] =	kill[I] =
if I is x = addr y :	gen[I] =	kill[I] =
if I is if (x) :	gen[I] =	kill[I] =
if I is return x :	gen[I] =	kill[I] =
if I is $x = f(y_1,\ldots, y_n)$:	gen[I] =	kill[I] =

- Transfer functions $F_I(X) = (X - kill[I]) \cup gen[I]$

AE: Transfer Functions

- Define kill/gen for each type of instruction

if I is x = y OP z :	gen[I] = {y OP z}	kill[I] = {E | x∈E}
if I is x = OP y :	gen[I] = {OP z}	kill[I] = {E | x∈E}
if I is x = y :	gen[I] = {}	kill[I] = {E | x∈E}
if I is x = addr y :	gen[I] = {}	kill[I] = {E | x∈E}
if I is if (x) :	gen[I] = {}	kill[I] = {}
if I is return x :	gen[I] = {}	kill[I] = {}
if I is $x = f(y_1,\ldots, y_n)$:	gen[I] = {}	kill[I] = {E | x∈E}

- Transfer functions $F_I(X) = (X - kill[I]) \cup gen[I]$

Available Expressions

- Lattice: $(2^E, \subseteq)$; has finite height
- Meet is set intersection, top element is E
- Is a forward dataflow analysis
- Dataflow equations:

  $out[I] = F_B(in[I])$, for all B

  $in[B] = \cap \{out[B'] \mid B' \in pred(B)\}$, for all B

  $in[B_s] = X_0$

- Transfer functions: $F_I(X) = (X - kill[I]) \cup gen[I]$

  - are monotonic and distributive

- Iterative solving of dataflow equation:

  - terminates
  - computes MOP solution

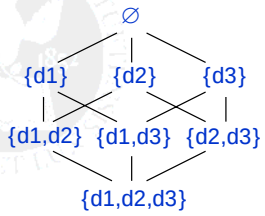
Problem 3: Reaching Definitions

- 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 $(D, \sqsubseteq)$, where:
  - $L = 2^D$ (power set of D)
  - Partial order $\sqsubseteq$ is set inclusion: $\supseteq$

    $S_1 \sqsubseteq S_2$ 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: $\varnothing$ (empty set)



RD: Dataflow Equations

- Equations:
  $out[I] = F_B(in[I])$, for all B
  $in[B] = \cup \{out[B'] \mid B' \in pred(B)\}$, for all B
  $in[B_s] = X_0$
- 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:
  $F_I(X) = (X - kill[I]) \cup gen[I]$
  where:
  kill[I] = definitions "killed" by I
  gen[I] = 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:
  gen[I] = {d} kill[I] = {d' | d' defines x}
- If I is not a definition:
  gen[I] = {} kill[I] = {}
- Transfer functions $F_I(X) = (X - kill[I]) \cup gen[I]$
- They are monotonic and distributive
  - For each F_I, kill[I] and 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 $\varnothing$
- Is a forward dataflow analysis
- Dataflow equations:
  $out[I] = F_B(in[I])$, for all B
  $in[B] = \cup \{out[B'] \mid B' \in pred(B)\}$, for all B
  $in[B_s] = X_0$
- Transfer functions: $F_I(X) = (X - kill[I]) \cup gen[I]$
  - 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)