CS412/413

Introduction to Compilers

Radu Rugina

Lecture 26: Standard Dataflow Analyses

03 Apr 06

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:

  $FP \sqsubseteq MFP \sqsubseteq MOP \sqsubseteq IDEAL$
- 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,z} may be live at program point p

LV: Dataflow Equations

- Equations:

  $\texttt{in}[n] = (\texttt{out}[n] - \texttt{def}[n]) \cup \texttt{use}[n]$, for all n

  $\texttt{out}[n] = \cup \{\texttt{in}[n'] \mid n' \in \texttt{succ}(n)\}$, for all n

  $\texttt{out}[n_0] = d_0$
- Where:

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

  use[n] = set of variables used (read) by n
- Meaning of transfer function:

  "A variable is live before a node if the node uses it, or if the variable is live after and the node doesn't define it"
- Meaning of union operator:

  "A variable is live at the end of node n if it is live at the beginning of one of its successor nodes"

LV: Monotonicity

- Are transfer functions: $F_n(X) = (X - def[n]) \cup use[n]$ monotonic?
- Observation: def[n] and use[n] are constant: they do not depend on the current dataflow information X.
- Because def[n] is constant, $G(X) = X - def[n]$ is monotonic:

  $X1 \supseteq X2$ implies $X1 - def[n] \supseteq X2 - def[n]$
- Because use[n] is constant, $H(Y) = Y \cup use[n]$ is monotonic:

  $Y1 \supseteq Y2$ implies $Y1 \cup use[n] \supseteq Y2 \cup use[n]$
- Put pieces together: $F_n(X)$ is monotonic

  $X1 \supseteq X2$ implies

  $(X1 - def[n]) \cup use[n] \supseteq (X2 - def[n]) \cup use[n]$

LV: Distributivity

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

Live Variables: Summary

- Lattice: $(2^V, \supseteq, \cup, \emptyset)$, has finite height
- Meet is set union, top is the empty set
- Is a backward dataflow analysis
- Dataflow equations:
  $in[n] = (out[n] - def[n]) \cup use[n]$, for all n
  $out[n] = \cup \{in[n'] \mid n' \in succ(n)\}$, for all n
  $out[n_0] = d_0$
- Transfer functions are monotonic and distributive
- Iterative solution to dataflow equations:
  - 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
- 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

AE: Dataflow Equations

- Equations:
  $out[n] = F_n(in[n])$, for all n
  $in[n] = \cap \{out[n'] \mid n' \in pred(n)\}$, for all n
  $in[n_0] = d_0$
- Meaning of intersection meet operator:
  "An expression is available at entry of node n if it is available at the exit of all predecessors"

AE: Transfer Functions

- General form of transfer functions:
  $F_n(X) = (X - kill[n]) \cup gen[n]$
  where:
  kill[n] = expressions "killed" by n
  gen[n] = new expressions "generated" by n
- Meaning of transfer functions: "Expressions available after node n include: 1) expressions available before n, not killed by n, and 2) expressions generated by n"

Available Expressions: Summary

- Lattice: $(2^E, \subseteq, \cap, 2^E)$; has finite height
- Meet is set intersection, top element is entire set
- Is a forward dataflow analysis
- Dataflow equations:
  $out[n] = F_n(in[n])$, for all n
  $in[n] = \cap \{out[n'] \mid n' \in pred(n)\}$, for all n
  $in[n_0] = d_0$
- Transfer functions: $F_I(X) = (X - kill[n]) \cup gen[n]$
  - 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

RD: Dataflow Equations

- Equations:

```
out[n] = (in[n] – kill[n] ) ∪ gen[n], for all n
in[n] = ∪ {out[n'] | n'∈pred(n)}, for all n
in[n0]  = d0
```

- Meaning of intersection meet operator:
  "A definition reaches the entry of node n if it reaches the exit of at least one of its predecessor nodes"
- Meaning of transfer functions: "Reaching definitions after node n include: 1) reaching definitions before n, not killed by n, and 2) reaching definitions generated by n"

Reaching Definitions: Summary

- Lattice: $(2^D, \supseteq, \cup, \emptyset)$; has finite height
- Meet is set union, top element is $\emptyset$
- Is a forward dataflow analysis
- Transfer functions are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution

Efficient 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 sparse subsets
- 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
- Let N = set of integer numbers
- The lattice is a map from V to N
- Construct the lattice starting from a lattice for N

Constant Folding Lattice

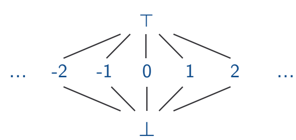
- Second try: lattice $(N \cup \{\top, \bot\}, \leq)$
  - Where $\bot \leq m$, for all $m \in N$
  - And $m \leq \top$, for all $m \in N$
  - Is complete!
- Meaning:
  - $v = \top$: don't know if v is constant
  - $v = \bot$: v is not constant

Constant Folding Lattice

- Second try: lattice $(N \cup \{\top, \bot\}, \leq)$
  - Where $\bot \leq m$, for all $m \in N$
  - And $m \leq \top$, for all $m \in N$
  - Is complete!
- Problem:
  - Is incorrect for constant folding
  - Meet of two constants $c \neq d$ is min(c,d)
  - Meet of different constants should be $\bot$
- Another problem: has infinite height …

Constant Folding Lattice

- Solution: flat lattice $L = (N \cup \{\top, \bot\}, \sqsubseteq)$
  - Where $\bot \sqsubseteq m$, for all $n \in N$
  - And $m \sqsubseteq \top$, 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 node n:

  $F_n(X) = (X - kill[n]) \cup gen[n]$
- Dataflow information X is a map from V to $N \cup \{\top, \bot\}$
  - Represent it as a set of pairs $(\texttt{var} \mapsto \texttt{m})$
  - Denote by $X[\texttt{var}] = \texttt{m}$ the value of var in this mapping
- If n is `v = c` (constant): $gen[n] = \{\texttt{v} \mapsto \texttt{c}\}$ $kill[n] = \{\texttt{v} \mapsto \_\}$
- If n is `v = u+w`: $gen[n] = \{\texttt{v} \mapsto \texttt{e}\}$ $kill[n] = \{\texttt{v} \mapsto \_\}$

  where $e = X[\texttt{u}] + X[\texttt{w}]$, if $X[\texttt{u}]$ and $X[\texttt{w}]$ are not $\top, \bot$

  $e = \bot$, if $X[\texttt{u}] = \bot$ or $X[\texttt{w}] = \bot$

  $e = \top$, if $X[\texttt{u}] = \top$ or $X[\texttt{w}] = \top$

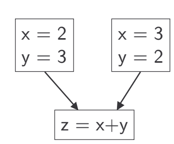
CF: Transfer Functions

- Transfer function for node n:

  $F_n(X) = (X - kill[n]) \cup gen[n]$
- Here gen[n] is not constant, it depends on X
- Exercise: prove that transfer functions are monotonic
- However, transfer functions are not distributive

CF: Distributivity

- Example:



- MFP and MOP yield different solutions

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: $\sqcap = \cup$, $\top = \emptyset$
  - 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: $\sqcap = \cap$, $\top = S$
  - Hence, initialize info to the whole set
  - Examples: available expressions