

## 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

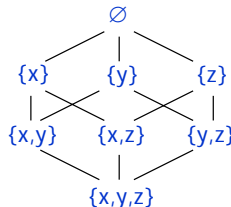
- Apply dataflow framework to several analysis problems:
  - Live variable analysis
  - Available expressions
  - Reaching definitions
  - Constant folding
- Discuss:
  - Implementation issues
  - Classification of dataflow analyses

## 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
- 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:  $\supseteq$
  - $S_1 \sqsubseteq S_2$  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:  $\cup$   
(set union:  $out[B]$  is union of  $in[B']$ , for all  $B' \in succ(B)$ )
- Top element:  $\emptyset$   
(empty set)
- Smaller sets of live variables = more precise analysis
- All variables may be live = least precise



## LV: Dataflow Equations

- Equations:
  - $in[B] = F_B(out[B])$ , for all B
  - $out[B] = \cup \{in[B'] \mid B' \in succ(B)\}$ , 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

- 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, not written by I, and 2) variables used by I"

CS 412/413 Spring 2005

Introduction to Compilers

7

## 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 <b>if</b> (x) :	$\text{use}[I] = \{x\}$	$\text{def}[I] = \{\}$
if I is <b>return</b> x :	$\text{use}[I] = \{x\}$	$\text{def}[I] = \{\}$
if I is $x = f(y_1, \dots, y_n)$ :	$\text{use}[I] = \{y_1, \dots, y_n\}$	$\text{def}[I] = \{x\}$
- 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

CS 412/413 Spring 2005

Introduction to Compilers

8

## 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:  
 $X1 \supseteq X2$  implies  $X1 - \text{def}[I] \supseteq X2 - \text{def}[I]$
- Because  $\text{use}[I]$  is constant,  $Y \cup \text{use}[I]$  is monotonic:  
 $Y1 \supseteq Y2$  implies  $Y1 \cup \text{use}[I] \supseteq Y2 \cup \text{use}[I]$
- Put pieces together:  $F_I(X)$  is monotonic  
 $X1 \supseteq X2$  implies  
 $(X1 - \text{def}[I]) \cup \text{use}[I] \supseteq (X2 - \text{def}[I]) \cup \text{use}[I]$

CS 412/413 Spring 2005

Introduction to Compilers

9

## 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:  
 $(X1 \cup X2) - \text{def}[I] = (X1 - \text{def}[I]) \cup (X2 - \text{def}[I])$   
 because:  $(a \cup b) - c = (a - c) \cup (b - c)$
- Since  $\text{use}[I]$  is constant:  $Y \cup \text{use}[I]$  is distributive:  
 $(Y1 \cup Y2) \cup \text{use}[I] = (Y1 \cup \text{use}[I]) \cup (Y2 \cup \text{use}[I])$   
 because:  $(a \cup b) \cup c = (a \cup c) \cup (b \cup c)$
- Put pieces together:  $F_I(X)$  is distributive  
 $F_I(X1 \cup X2) = F_I(X1) \cup F_I(X2)$

CS 412/413 Spring 2005

Introduction to Compilers

10

## 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:  
 $\text{in}[B] = F_B(\text{out}[B])$ , for all B  
 $\text{out}[B] = \cup \{\text{in}[B'] \mid B' \in \text{succ}(B)\}$ , 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

CS 412/413 Spring 2005

Introduction to Compilers

11

## 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:  $\sqsubseteq$   
 $S_1 \sqsubseteq S_2$  iff  $S_1 \subseteq S_2$

CS 412/413 Spring 2005

Introduction to Compilers

12

## AE: The Lattice

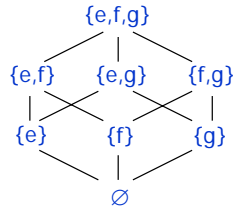
- Consider set of expressions =  $\{x^*z, x+y, y-z\}$
- Denote  $e = x^*z, f=x+y, g=y-z$

- 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 variables = more precise analysis
- No available expressions = least precise



CS 412/413 Spring 2005

Introduction to Compilers

13

## AE: Dataflow Equations

- Equations:

$$\text{out}[I] = F_B(\text{in}[I]), \text{ for all } B$$

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

$$\text{in}[B_s] = X_0$$

- Meaning of intersection meet operator:  
"An expression is available at entry of block B if it is available at exit of all predecessor nodes"

CS 412/413 Spring 2005

Introduction to Compilers

14

## 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:

$\text{kill}[I]$  = expressions "killed" by I

$\text{gen}[I]$  = 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"

CS 412/413 Spring 2005

Introduction to Compilers

15

## AE: Transfer Functions

- Define kill/gen for each type of instruction

$$\text{if } I \text{ is } x = y \text{ OP } z : \text{gen}[I] = \{y \text{ OP } z\} \quad \text{kill}[I] = \{E \mid x \in E\}$$

$$\text{if } I \text{ is } x = \text{OP } y : \text{gen}[I] = \{\text{OP } y\} \quad \text{kill}[I] = \{E \mid x \in E\}$$

$$\text{if } I \text{ is } x = y : \text{gen}[I] = \{y\} \quad \text{kill}[I] = \{E \mid x \in E\}$$

$$\text{if } I \text{ is } x = \text{addr } y : \text{gen}[I] = \{y\} \quad \text{kill}[I] = \{E \mid x \in E\}$$

$$\text{if } I \text{ is if } (x) : \text{gen}[I] = \{y\} \quad \text{kill}[I] = \{y\}$$

$$\text{if } I \text{ is return } x : \text{gen}[I] = \{x\} \quad \text{kill}[I] = \{y\}$$

$$\text{if } I \text{ is } x = f(y_1, \dots, y_n) : \text{gen}[I] = \{f(y_1, \dots, y_n)\} \quad \text{kill}[I] = \{E \mid x \in E\}$$

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

- ... how about  $x = x \text{ OP } y$ ?

CS 412/413 Spring 2005

Introduction to Compilers

16

## 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:

$$\text{out}[I] = F_B(\text{in}[I]), \text{ for all } B$$

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

$$\text{in}[B_s] = X_0$$

- 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

CS 412/413 Spring 2005

Introduction to Compilers

17

## 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:

$$L = 2^D \text{ (power set of } D)$$

$$\text{Partial order } \subseteq \text{ is set inclusion: } \supseteq$$

$$S_1 \subseteq S_2 \text{ iff } S_1 \supseteq S_2$$

CS 412/413 Spring 2005

Introduction to Compilers

18

## RD: The Lattice

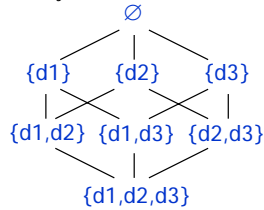
- Consider set of expressions = {d1, d2, d3}  
where d1: x = y, d2: x=x+1, d3: z=y-x

- Partial order:  $\supseteq$
- Set D is finite implies  
lattice has finite height

- Meet operator:  $\cup$   
(set union)

- Top element:  $\emptyset$   
(empty set)

- Smaller sets of reaching definitions = more precise analysis
- All definitions may reach current point = least precise



CS 412/413 Spring 2005

Introduction to Compilers

19

## RD: Dataflow Equations

- Equations:

$$\text{out}[I] = F_B(\text{in}[I]), \text{ for all } B$$

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

$$\text{in}[B_s] = X_0$$

- 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"

CS 412/413 Spring 2005

Introduction to Compilers

20

## 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:

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"

CS 412/413 Spring 2005

Introduction to Compilers

21

## RD: Transfer Functions

- Define kill/gen for each type of instruction

- If I is a definition d that defines x:

$$\text{gen}[I] = \{d\}$$

$$\text{kill}[I] = \{d' \mid d' \text{ defines } x\}$$

- If I is not a definition:

$$\text{gen}[I] = \{\}$$

$$\text{kill}[I] = \{\}$$

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

- They are monotonic and distributive

- For each  $F_I$ , kill[I] and gen[I] are constants: they don't depend on input information X

CS 412/413 Spring 2005

Introduction to Compilers

22

## Reaching Definitions: Summary

- Lattice:  $(2^D, \supseteq)$ ; has finite height
- Meet is set union, top element is  $\emptyset$
- Is a forward dataflow analysis

- Dataflow equations:

$$\text{out}[I] = F_B(\text{in}[I]), \text{ for all } B$$

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

$$\text{in}[B_s] = X_0$$

- 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

CS 412/413 Spring 2005

Introduction to Compilers

23

## 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.

CS 412/413 Spring 2005

Introduction to Compilers

24

## 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

CS 412/413 Spring 2005

Introduction to Compilers

25

## 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,  $nvar = |V|$
- Let  $N$  = set of integer numbers
- 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!  
... why?

CS 412/413 Spring 2005

Introduction to Compilers

26

## Constant Folding Lattice

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



CS 412/413 Spring 2005

Introduction to Compilers

27

## Constant Folding Lattice

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



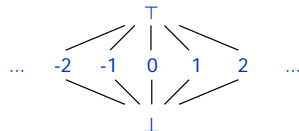
CS 412/413 Spring 2005

Introduction to Compilers

28

## Constant Folding Lattice

- **Solution:** flat lattice  $L = (N \cup \{\top, \perp\}, \sqsubseteq)$ 
  - Where  $\perp \sqsubseteq n$ , for all  $n \in N$
  - And  $n \sqsubseteq \top$ , for all  $n \in N$
  - And distinct integer constants are not comparable



- **Note:** meet of any two distinct numbers is  $\perp$ !

CS 412/413 Spring 2005

Introduction to Compilers

29

## Constant Folding Lattice

- Denote  $N^* = N \cup \{\top, \perp\}$
- Use flat lattice  $L = (N^*, \sqsubseteq)$
- **Constant folding lattice:**  $L' = (V \rightarrow N^*, \sqsubseteq_c)$
- Where partial order on  $V \rightarrow N^*$  is defined as:
  - $X \sqsubseteq_c Y$  iff for each variable  $v$ :  $X(v) \sqsubseteq Y(v)$
- Can represent a function in  $V \rightarrow N^*$  as a set of assignments:  $\{v_1=c_1\}, \{v_2=c_2\}, \dots, \{v_n=c_n\}$

CS 412/413 Spring 2005

Introduction to Compilers

30

## CF: Transfer Functions

- Transfer function for instruction I:  
 $F_I(X) = (X - \text{kill}[I]) \cup \text{gen}[I]$   
 where:  
 $\text{kill}[I]$  = constants "killed" by I  
 $\text{gen}[I]$  = constants "generated" by I
- $X[v] = c \in N^*$  if  $\{v=c\} \in X$
- If I is  $v = c$  (constant):  $\text{gen}[I] = \{v=c\}$   $\text{kill}[I] = \{v\} \times N^*$
- If I is  $v = u+w$ :  $\text{gen}[I] = \{v=e\}$   $\text{kill}[I] = \{v\} \times N^*$   
 where  $e = X[u] + X[w]$ , if  $X[u]$  and  $X[w]$  are not  $\perp, \perp$   
 $e = \perp$ , if  $X[u] = \perp$  or  $X[w] = \perp$   
 $e = \top$ , if  $X[u] = \top$  or  $X[w] = \top$

CS 412/413 Spring 2005

Introduction to Compilers

31

## 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 (easy to prove)
- ... but are transfer functions distributive?

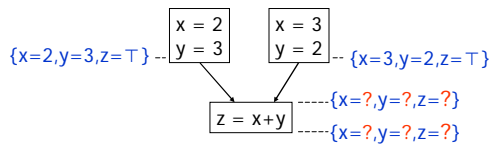
CS 412/413 Spring 2005

Introduction to Compilers

32

## CF: Distributivity

- Example:



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

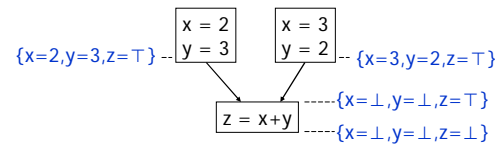
CS 412/413 Spring 2005

Introduction to Compilers

33

## CF: Distributivity

- Example:



- Dataflow result (MFP) at the end:  $\{x=\perp, y=\perp, z=\perp\}$
- MOP solution at the end?

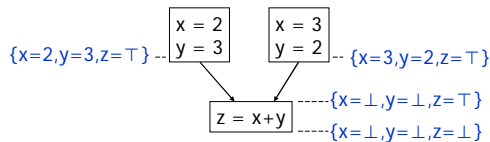
CS 412/413 Spring 2005

Introduction to Compilers

34

## CF: Distributivity

- Example:



- Dataflow result (MFP) at the end:  $\{x=\perp, y=\perp, z=\perp\}$
- MOP solution at the end:  $\{x=\perp, y=\perp, z=5\}$  !

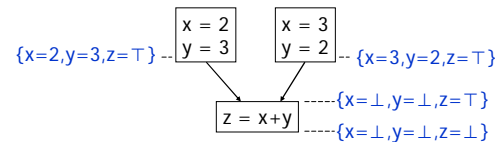
CS 412/413 Spring 2005

Introduction to Compilers

35

## CF: Distributivity

- Example:



- Reason for MOP  $\neq$  MFP:  
**transfer function F of  $z=x+y$  is not distributive!**  
 $F(X1 \sqcap X2) \neq F(X1) \sqcap F(X2)$   
 where  $X1 = \{x=2, y=3, z=\top\}$  and  $X2 = \{x=3, y=2, z=\top\}$

CS 412/413 Spring 2005

Introduction to Compilers

36

## 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:  $\Pi = \cap, \top = S$
  - Hence, initialize info to the whole set
  - **Examples:** available expressions