

CSc 553

Principles of Compilation

28 : Optimization III

Department of Computer Science
University of Arizona

collberg@gmail.com

Copyright © 2011 Christian Collberg

Introduction

Computing Data-Flow Info.

- There are two principal methods of solving data-flow problems:
 - 1 Let `gen`, `kill`, `in`, `out` be AST attributes and the data-flow equations attribute evaluation rules. We'll look at this later.
 - 2 Treat data-flow equations as recurrences, and iterate over the set of equations until a solution is found.
- Sets are stored as bit-vectors, with one element for each possible object.

$$\begin{aligned} \text{in}[B1] &= \{d_3, d_5, d_7\} \\ &\equiv \begin{array}{c|c|c|c|c|c|c|c} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 \end{array} \end{aligned}$$

Data-flow Analysis by Iteration

Reaching Definitions — Equations

$$\begin{aligned}\text{out}[B] &= \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \\ \text{in}[B] &= \bigcup_{\text{preds } P \text{ of } B} \text{out}[P]\end{aligned}$$

- A definition $d : a := b + c$ reaches a use of a at point p , if the value given to a at d could be used at p .
- $\text{gen}[B]$ is the set of definitions generated within B , that reach the end of B .
- $\text{kill}[B]$ is the set of definitions outside B , killed by definitions within B .
- $\text{in}[B]$ is the set of definitions valid at the entrance to B , $\text{out}[B]$ is those valid at the exit of B .
- The equations for in and out are valid for each basic block.

Iterative Algorithms I

- 1 Compute `gen` and `kill` for each block.
- 2 Set up the $2n$ in- and out-equations for the n basic blocks, and set $\text{in}[B] = \text{out}[B] = \{\}$ for each block B .
- 3 Repeat until no more changes:
 - For each block B eval. $\text{in}[B]$ & $\text{out}[B]$.

Formal Algorithm:

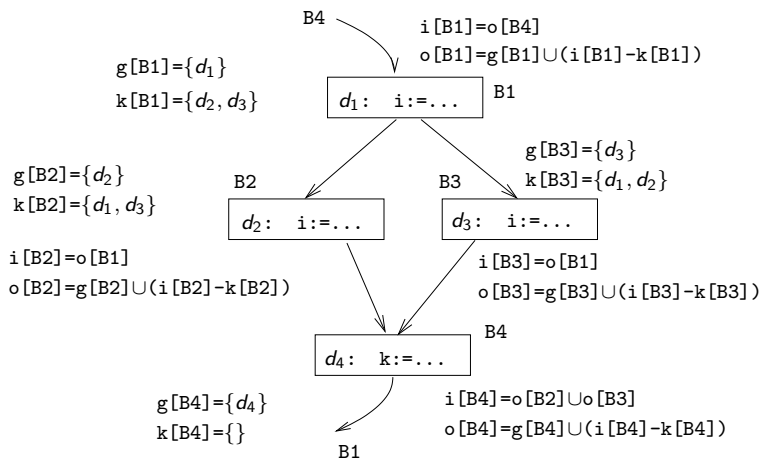
```
FOR each block  $B$  DO  
     $\text{out}[B] := \text{in}[B] := \{\}$ ;  
END;  
WHILE any  $\text{out}[B]$  has changed DO  
    FOR each block  $B$  DO  
         $\text{in}[B] := \bigcup_{\text{preds } P \text{ of } B} \text{out}[P]$   
         $\text{out}[B] := \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B])$   
    END;  
END
```

Iterative Alg. Example I (a)

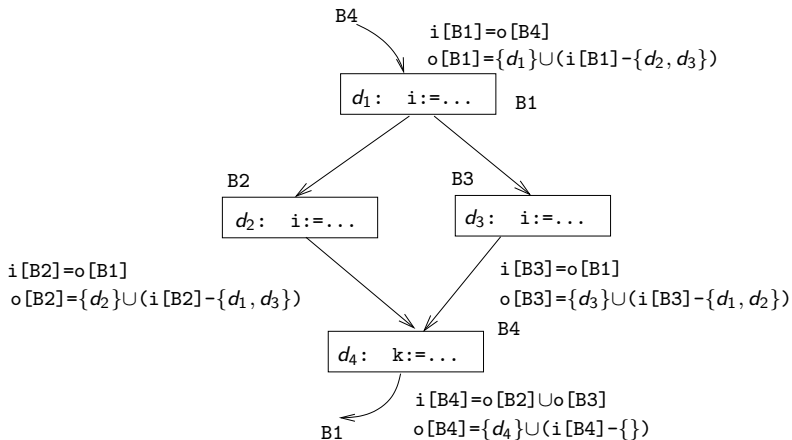
```
REPEAT
   $d_1$ :  $i := \dots$ ;
      IF ...THEN
 $d_2$ :    $i := \dots$ 
      ELSE
 $d_3$ :    $i := \dots$ 
      ENDIF;
 $d_4$ :  $k := \dots$ 
UNTIL ...;
```

- Start by setting up the $2n$ in- and out-equations (slide (b)).
- Simplify the example by inlining `gen` and `kill` into the equations for `in` and `out` (slide (c)).
- Visit each block in turn (we use numerical order, B1, B2, B3, B4) and evaluate `in` and `out` (slides (d)–(g)).

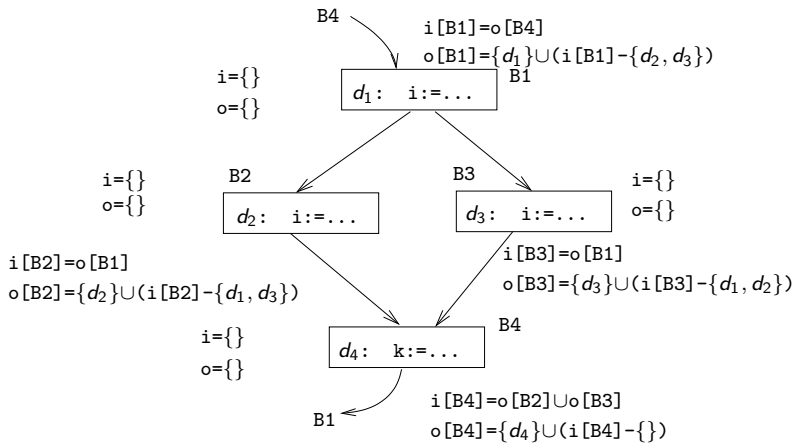
Iterative Alg. Example I (b)



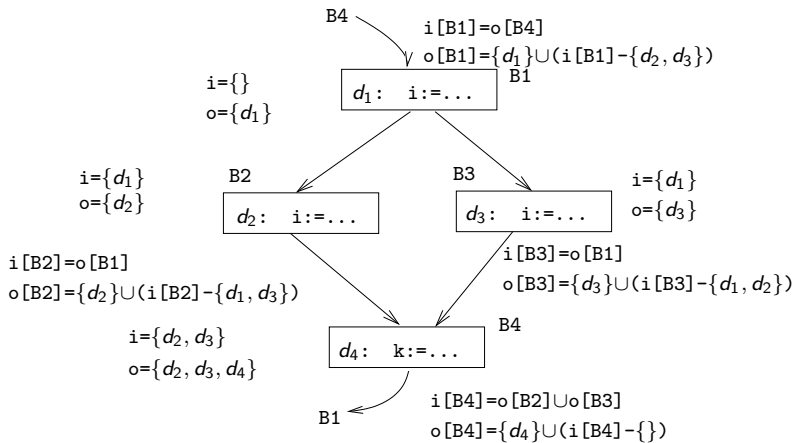
Iterative Alg. Example I (c) – Simplified



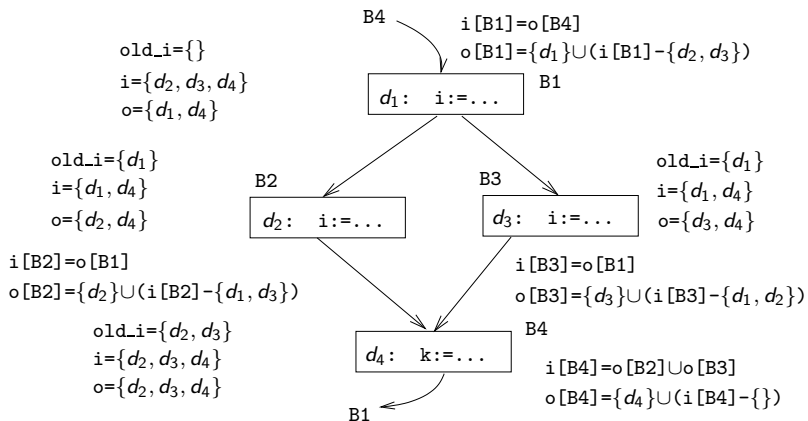
Iterative Alg. Example I (d) – Initialized



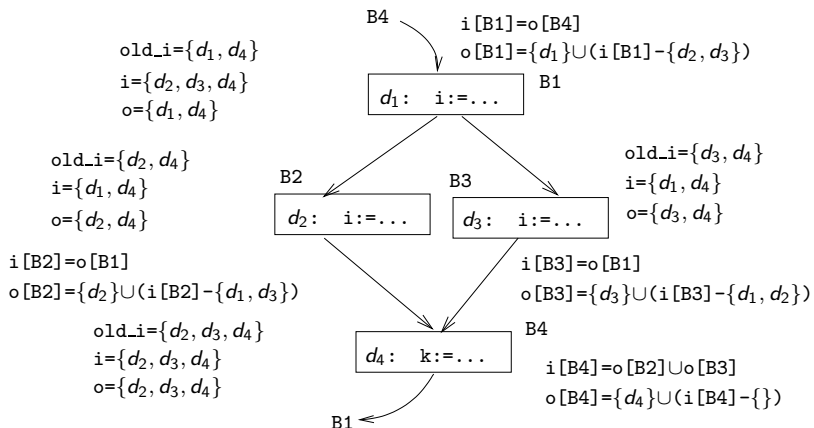
Iterative Alg. Example I (e) – 1st Iteration



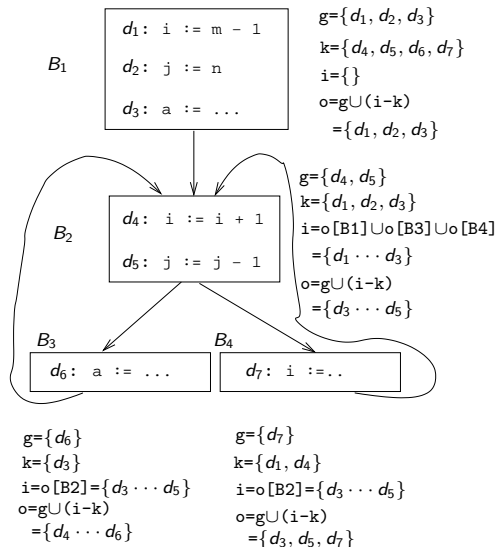
Iterative Alg. Example I (f) – 2nd Iteration



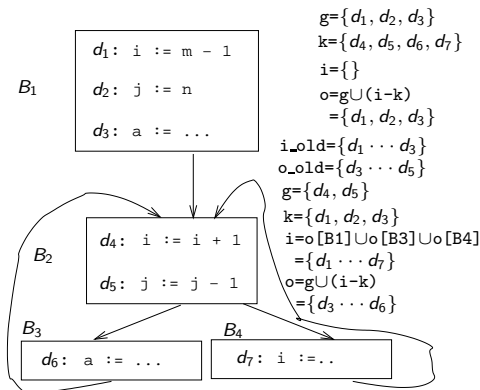
Iterative Alg. Example I (g) – 3rd Iteration



Example II (a) – 1st Iteration



Example II (b) – 2nd Iteration



$$g = \{d_6\} \quad k = \{d_3\}$$

$$i = o[B2] = \{d_3 \dots d_6\}$$

$$o = g \cup (i - k)$$

$$= \{d_4 \dots d_6\}$$

$$o_old = \{d_4 \dots d_6\}$$

$$i_old = \{d_3 \dots d_5\}$$

$$g = \{d_7\} \quad k = \{d_1, d_4\}$$

$$i = o[B2] = \{d_3 \dots d_6\}$$

$$o = g \cup (i - k)$$

$$= \{d_3, d_5 \dots d_7\}$$

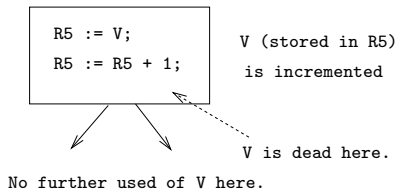
$$o_old = \{d_3, d_5, d_7\}$$

$$i_old = \{d_3 \dots d_5\}$$

Live Variable Analysis

Live Variable Analysis I

- For each definition/use of a variable V , **Global Live Variable Analysis** answers the question
“Could the value of V computed/used here be used further on in the program?”
- If a variable V is stored in a register $R5$ and V is dead at the end of the block, then we don't have to store $R5$ back into V .
- Assignments to dead variables can be removed.



Live Variable Analysis II

- in [B]** Variables live on entrance to B.
- out [B]** Variables live on exit from B.
- def [B]** Variables assigned values in B before the variable is used:

```
B := ... C ...;
C := ...;
... := ... B ...;
```

```
def={ B }
C is not in def since
its value is being
used before C is
assigned to.
```

- use [B]** Variables whose values are used before being assigned to:

```
B := ... C ...;
C := ...;
... := ... B ...;
```

```
use={ C }
B is not in use since
it is assigned a value
before being used.
```

Live Variable Analysis III

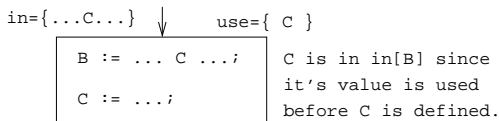
_____ Data-Flow (Equations): _____

$$\text{in}[B] = \text{use}[B] \cup (\text{out}[B] - \text{def}[B])$$

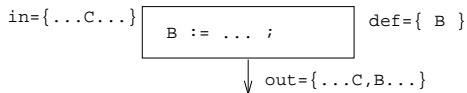
_____ Data-Flow (English): _____

- V is live at the entrance to B if

- 1 it is being used before it's defined (i.e. $V \in \text{use}[B]$)



- 2 it is live at the exit of the block, and it is not defined within the block (i.e. $V \in \text{out}[B]$ and $V \notin \text{def}[B]$)



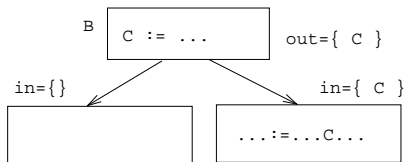
Live Variable Analysis IV

_____ Data-Flow (Equations): _____

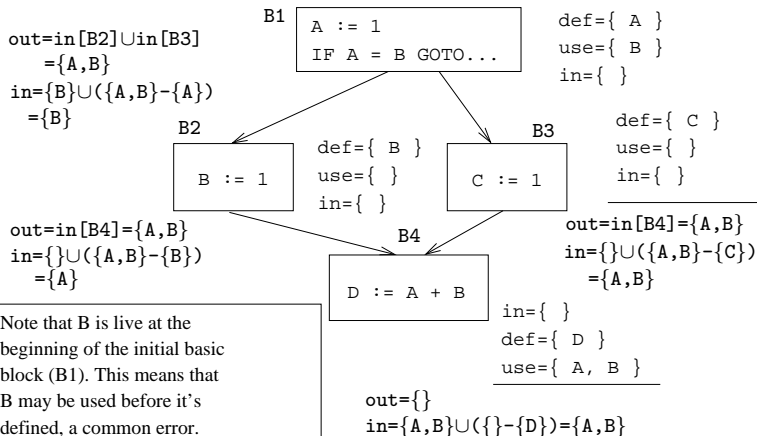
$$\text{out}[B] = \bigcup_{\substack{\text{successors} \\ S \text{ of } B}} \text{in}[S]$$

_____ Data-Flow (English): _____

- A variable V is live coming out of B if it is live going into any one of B 's successors.



Live Variable Analysis V – Example



Live Variables VI – Example

$out[B4] = \{\}$ since out is the union of all of $B4$'s successor's in , and $B4$ doesn't have any successors.

$in[B4] = \{\}$ because both A & B are live coming in to $B4$, i.e. their values will be used before they are assigned new values.

$out[B3] = in[B4] = \{A, B\}$ because the values of A and B will be used in $B3$'s successor block, $B4$. Note that since $C \notin out[B3]$ C 's value is *dead* and the assignment $C := 1$ can be removed.

$out[B1] = \{A\} \cup \{A, B\} = \{A, B\}$ since if we take the left branch (through $B2$) A will be used further on, and if we take the right branch (through $B3$) both A and B will have a future use.

$in[B1] = \{B\}$ since B 's value is used but not defined in B .

Classifying Data-Flow Problems

Classification — Forward vs. Backward Flow

- Data-flow problems can be classified according to the direction of flow:

Forward-flow problems: Data flows from the initial block to the end block.

- Out-sets are computed from In-sets within basic blocks,
- In-sets are computed from Out-sets across basic blocks.

Backward-flow problems: Data flows from the end block to the initial block.

- In-sets are computed from Out-sets within basic blocks,
- Out-sets are computed from In-sets across basic blocks.

Forward vs. Backward Flow — Equations

	Forward-Flow	Backward-Flow
Any Path	$o_B = g_B \cup (i_B - k_B)$ $i_B = \bigcup_{b \in P(B)} o_b$	$i_B = g_B \cup (o_B - k_B)$ $o_B = \bigcup_{b \in S(B)} i_b$
All Paths	$o_B = g_B \cup (i_B - k_B)$ $i_B = \bigcap_{b \in P(B)} o_b$	$i_B = g_B \cup (o_B - k_B)$ $o_B = \bigcap_{b \in S(B)} i_b$

- $P(B)$ = Predecessors of B , $S(B)$ = Successors of B .
- $i_B = \text{in}_B$, $o_B = \text{out}_B$, $g_B = \text{gen}_B$, $k_B = \text{kill}_B$, $o_b = \text{out}_b$, $i_b = \text{in}_b$.

Classification — Any- vs. All-Path

- We classify data-flow problems by the way they combine incoming information:

Any-path problems: All values coming in to a block are valid.
Use \cup .

All-path problems: Only values coming in to a block through every path are valid. Use \cap .

	Forward-Flow	Backward-Flow
Any Path	Reaching Definitions Uninitialized Variables	Live Variables Du-chains
All Paths	Available Expressions Copy Propagation	Very Busy Expressions

Summary

Summary I

- Read the Dragon Book: 608–611, 614–615, 618, 620–622, 624–633.
- With B blocks & bit-vectors of length V , iterative data-flow analysis is $\mathcal{O}(B^2 \times V)$ in the worst case.

Homework

Homework I

- Show each step of the iterative reaching definitions algorithm applied to the procedure body below:

```
K := 1; I := 2;
REPEAT
  IF I = 4 THEN
    A := K + 1;
  ELSE
    A := K + 2;
    I := I + A;
  ENDIF;
UNTIL I <= 10;
K := K + A;
```

Exam Problems

Exam Problem I (a) [07.430 '95]

- An expression E is *very busy* if – regardless of which path we take through the flow graph – E 's value will be used before it is killed. Example ($A+3$ is *very busy*):

```
(1)  BEGIN
(2)      IF expr THEN
(3)          V := A + 3;
(4)          R := K + 3;
(5)      ELSE
(6)          Z := A + 3;
(7)          K := 5;
(8)          L := K + 3;
(9)      END;
(10) END
```


Exam Problem I (b) [07.430 '95]

Data-Flow Equations:

- The data-flow equations for computing very busy expressions are:

$$\begin{aligned} \text{in}[B] &= \text{used}[B] \cup (\text{out}[B] - \text{killed}[B]) \\ \text{out}[B] &= \bigcap_{\substack{\text{successors} \\ S \text{ of } B}} \text{in}[S] \end{aligned}$$

Problems:

- Give an iterative pseudo-code routine for computing `in` and `out`.
- Is *very-busy expressions* a forward-flow or a backward-flow problem?
- Show the workings of the algorithm on the procedure body in the next slide:

Exam Problem I (c) [07.430 '95]

```
BEGIN
  X := 5;
  Y := 10;
  IF  $e_1$  THEN
    IF  $e_2$  THEN
      A := X * Y;
    ELSE
      B := 3;
      V := X * Y;
      X := 1;
    END;
  ELSE
    Y := 2;
    A := X * Y;
  END
END
```