Global Data-Flow Analysis

Introduction

- A number of optimization can be achieved by knowing various pieces of information that can be obtained only by examining the entire program
- If a variable A has a value 3 every time control reaches a certain point *p* for each use of A at *p*, then we can substitute 3 for each use of A at *p*
- This may require scanning the entire program
- Problem is *use definition* or *ud-chaining*
- It answers the question
- *“* Given that identifier A is used at point *p*, at what points could the value of A used at *p* have been defined?”

Introduction

- By a *use* of identifier A we mean any occurrence of A as an operand
- By a *definition* of A we mean either an assignment to A or the reading of a value of A (A to be the operand of a **param** statement)
- Difficulty when A is used as an Array
- What is a *point* in a program?
  - A position before or after any intermediate language statement

Introduction

- "A definition of a variable *reaches* a point *p* if there is a path in the flow graph from that definition to *p,* such that no other definitions of A appear on the path"

Reaching Definitions

- To determine definitions that can reach a given point in a program
  - we first assign distinct number to each definition
    - Since each definition is associated with a unique quadruple, the index of that quadruple will be used
  - Make a list of all definitions of A anywhere in the program

- Next step in ud-chaining is
- For each basic block B compute two sets
  - GEN[B] – the set of generated definitions (those definitions within block B that reach the end of the block)
  - KILL[B]- is the set of definitions outside of B that define identifiers that also have definitions within B

- Next step, the most complex is to compute for all blocks B, the set of IN[B]
- Let *u* be the statement in question using A
  - If there are definitions of A within block B prior to statement *u* , then the last such definition is the only definition of A reaching *u*
  - If there are no definitions of A within block B prior to statement *u* , then the last such definitions of A reaching *u* are these definitions of A that are in IN[B]

- To help compute IN we simultaneously compute OUT[B]
- OUT[B] – is the set of definitions reaching the point just after the last statement of B
- OUT[B] speaks of definitions everywhere, whereas GEN[B] speaks only of definitions inside B

DATA-Flow Equation

- 1)OUT[B]=IN[B]-KILL[B] U GEN[B]
- (2) IN[B]=U OUT[P]
- p is a predecessor of B
  - Rule 1 says that a definition of d reaches the end of block B iff either one of the following holds
  1. d is in IN[B]
  2. d is generated within B
  - Rule 2 says that a definition reaches the beginning of block B iff it reaches the end of one of its predecessors.

Algorithm: compute IN[B] and OUT[B]

- Input: a flow graph for which KILL[B] and GEN[B] have been computed for each block B
- Output: IN[B] and OUT[B] for each block B
- Method: we use an iterative approach, starting with the estimate IN[B]=ϕ for all B and converging to the desired values of IN and OUT.
- A boolean variable CHANGE is use to record the iteration on each pass through the blocks whether IN has changed.

Algorithm

Begin

1. For each block B do

   Begin

2. IN[B]:=ϕ;
3. OUT[B]:=GEN[B]

   End;

4. CHANGE:=true;
5. While CHANGE do

   Begin

6. CHANGE:=false;
7. For each block B do

   Begin

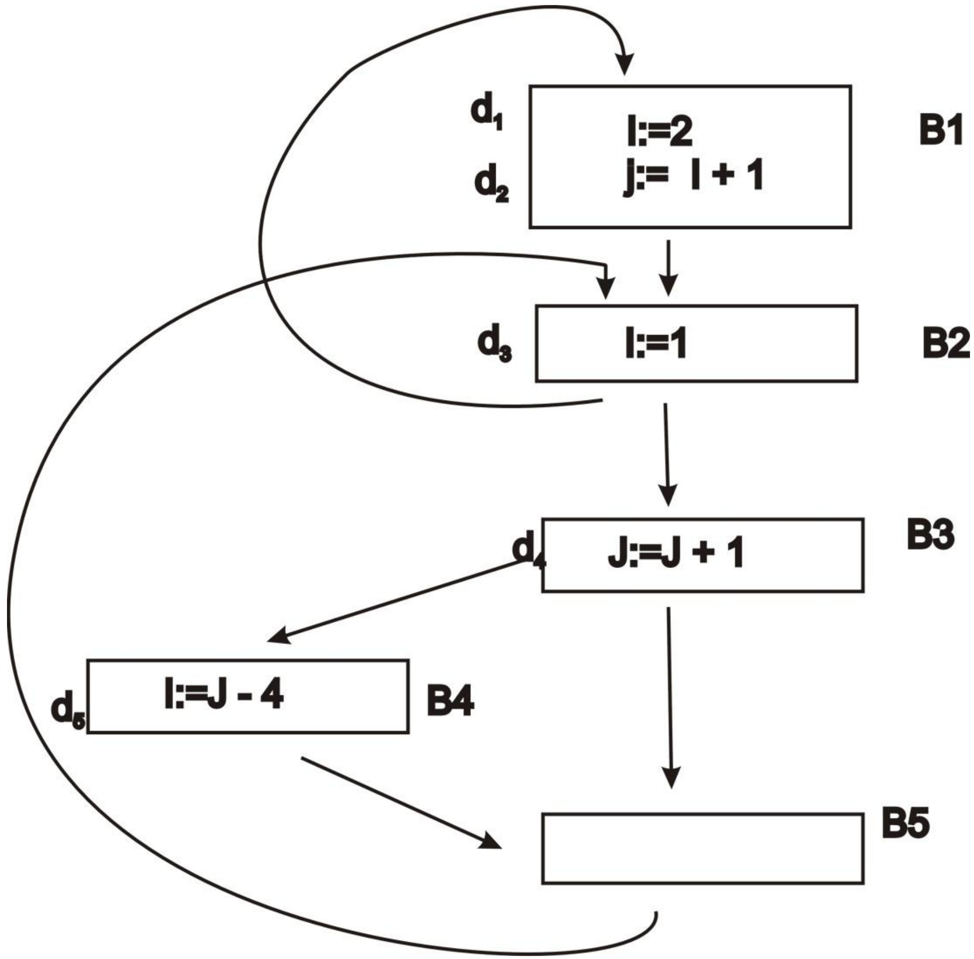
8. NEWIN:= U OUT[P];
9. If NEWIN≠IN[B] then CHANGE:=true;
10. IN[B]:=NEWIN;
11. OUT[B]:=IN[B] – KILL[B] U GEN[B]

   End

   End

End

A Flow graph



GEN and KILL

Block B	GEN[B]	Bit vector	KILL[B]	Bit vector
B1	$\{d_1,d_2\}$	11000	$\{d_3,d_4,d_5\}$	00111
B2	$\{d_3\}$	00100	$\{d_1\}$	10000
B3	$\{d_4\}$	00010	$\{d_2, d_5\}$	01001
B4	$\{d_5\}$	00001	$\{d_2, d_4\}$	01010
B5	ϕ	00000	ϕ	00000

	initial		Pass 1	
block	IN[B]	OUT[B]	IN[B]	OUT[B]
B1	00000	11000	00100	11000
B2	00000	00100	11000	01100
B3	00000	00010	01100	00110
B4	00000	00001	00110	00101
B5	00000	00000	00111	00111

- IN[B1]==IN[B2]=IN[B3]=IN[B4]=IN[B5]=00000
- OUT[B]=GEN[B]
- **B=B1**
- NEWIN=OUT[B2]=GEN[B2]=00100, SINCE B2 IS THE PREDECESSOR OF B1
- NEWIN≠IN[B1]
- IN[B1]=NEWIN
- OUT[B1]=IN[B1]-KILL[B1]+GEN[B1]
- OUT[B1]=00100 – 00111 + 11000=11000
  - Interpreted as A and (not B) or C

00100
00111=11000
And=00000
Or = 11000
= 11000

- **B=B2**
- NEWIN=OUT[B1] +OUT[B5]=GEN[B1]+GEN[5]=11000+00000=11000, SINCE B1 AND B5 ARE THE PREDECESSORS OF B2
- NEWIN≠IN[B2]
- IN[B2]=NEWIN
- OUT[B2]=IN[B2]-KILL[B2]+GEN[B2]
- OUT[B2]=11000-10000+00100=01100

11000
<u>10000=01111</u>
And=01000
<u>Or = 00100</u>
= 01100

	Pass 2		Pass 3	
block	IN[B]	OUT[B]	IN[B]	OUT[B]
B1	01100	11000	01111	11000
B2	11111	01111	11111	01111
B3	01111	00110	01111	00110
B4	00110	00101	00110	00101
B5	00111	00111	00111	00111

Computing ud-chains

- ud-chains can be computed from the reaching definition information
- In the flow graph there are three uses of names
  - d2 uses I, d4 and d5 uses J
  - The use of I at d2 in block B1 is preceded by a definition of I in B1, namely d1
  - Thus the ud-chain for I in d2 consists only of d1
- The use of J at d4 in B3 is not preceded by a definition of J in B3
  - Thus we consider IN[B3]=(d2,d3,d4,d5}
  - Of all these but d3 is a definition of J, so the ud chain of J in d4 is d2,d4,d5

Applications of ud-chains

- If there is only one definition of name A which reaches a point *p* and that definition is A:=3 then we know A has the value 3 at that point and we can substitute 3 for A if there is a use of A at that point
- Example
- $IN[B_5]=\{d_3,d_4,d_5\}$
  - Of the definitions only d3:I=1 is a definition. If we use I in B_5 then we can replace it by 1.

Applications of ud-chains

- we can determine whether a particular definition reaches anywhere at all
  - taking the logical OR of all the INs give us this information
- For a particular use of a variable A we can determine whether A is undefined at that point