Lecture 13 (Data flow analysis)
Analysis of Software Artifacts

Somesh Jha

Motivation

- global information about the program
- examples
  - does variable v effect a conditional?
  - can a certain code fragment be eliminated?
  - is a conditional always true?

What is it used for?

- constructing tests (we will see this later)
- program understanding
  - all the variables that effect a certain statement
  - *slicing*
- code optimization
  - an expression inside a loop always has the same value
  - dead code elimination

Data-flow analysis

- assume that your program can be represented as a graph

- each node in the program represents a *basic block*

- a *basic block* is a segment of code with no *conditional* statements in it

- edges in the graph represent conditional statements

- conditions corresponding to edges not shown

Real Languages

- most compilers do these kinds of analysis
- pointers make life harder
- if `x` is of type `(int *)` what does it point to?
- aliasing analysis
- OO languages can be handled quite easily
- concurrent programs are challenging

Generally conservative information

- if data-flow analysis claims that expression
- `x+y*y` has value 10 inside a loop
- it *will* have the value
- if analysis says no, it means *may be*

Reaching definitions analysis

- a *definition* of a variable `x` is a statement that assigns, or may assign, a value to `x`.

- a definition d reaches a point p if there is a path from d to p, such that d is not *killed* along that path.

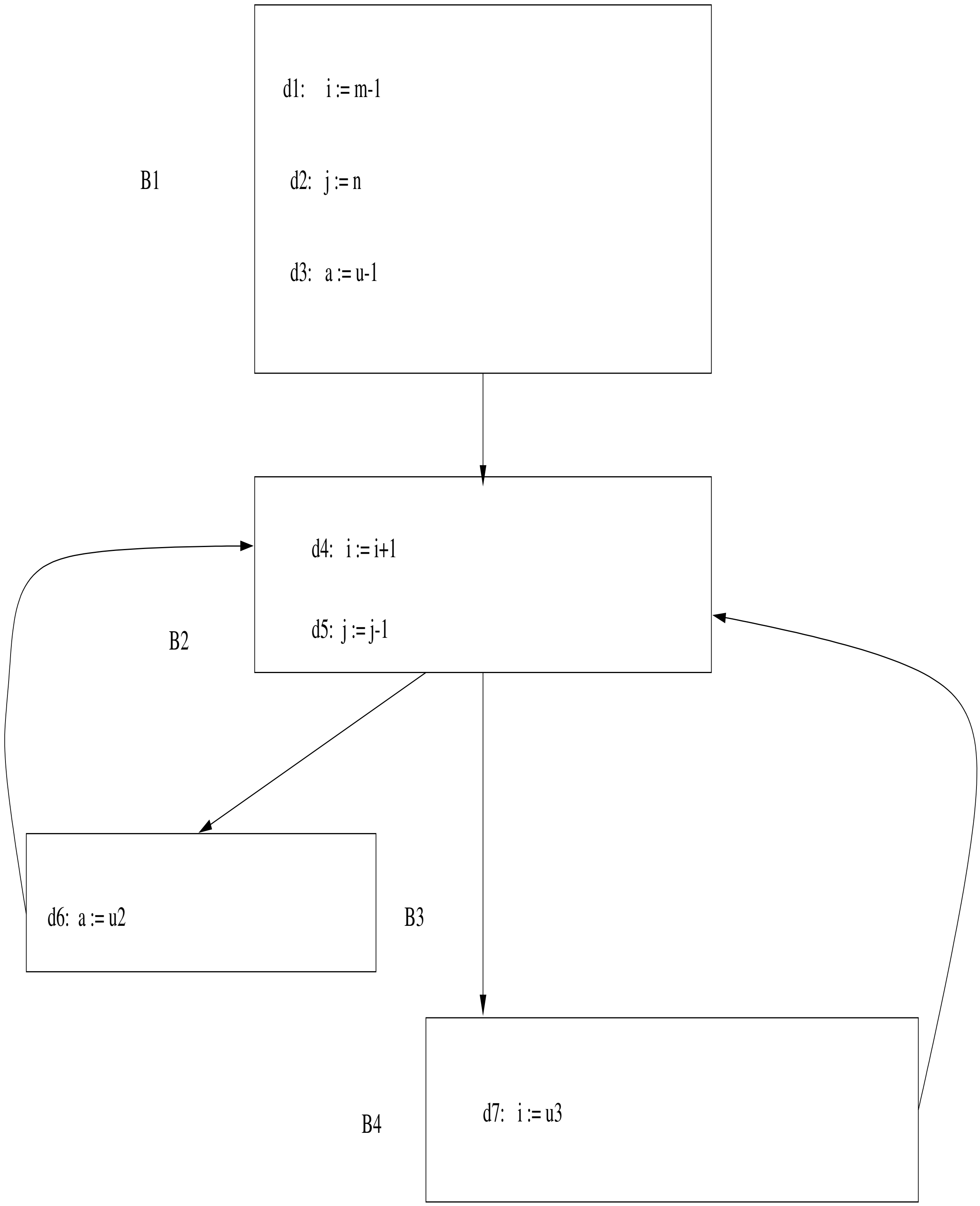

Figure 1: Sample program

Notation

- $out[B]$
  these are the definitions that are *alive* after executing basic-block B

- $in[B]$
  these are the definitions that are *alive* coming *into* the basic-block B

- $kill[B]$
  these are definitions that are *killed* while traversing block B

- $gen[B]$
  these are the definitions that are *generated* in the basic-block B

Notation (Contd)

- For block B_1

$$\begin{aligned} gen[B1] &= \{d1, d2, d3\} \\ kill[B1] &= \{d4, d5, d6, d7\} \end{aligned}$$

- For block B_2

$$\begin{aligned} gen[B2] &= \{d4, d5\} \\ kill[B2] &= \{d1, d2, d7\} \end{aligned}$$

- For block B_3

$$\begin{aligned} gen[B3] &= \{d6\} \\ kill[B3] &= \{d3\} \end{aligned}$$

- For block B_4

$$\begin{aligned} gen[B4] &= \{d7\} \\ kill[B4] &= \{d1, d4\} \end{aligned}$$

Fix-point equation

- initially, $out[B]$ is empty for each basic block B

- update $out[B]$ according to the following equations until you reach a fix-point:

$$
\begin{aligned}
in(B) &= \cup_{P \in pred(B)} out(P) \\
out(B) &= gen(B) \cup (in(B) - kill(B))
\end{aligned}
$$

- *fix-point:* for all basic blocks B the sets $in(B)$ and $out(B)$ do not change after an iteration

Pass 0

- sets are represented as bit-vectors
- each bit represents a definition
- computation for pass 0 is shown below:

block	*in*	*out*
$B1$	000 0000	111 0000
$B2$	000 0000	000 1100
$B3$	000 0000	000 0010
$B4$	000 0000	000 0001

Figure 2: Pass 0

Pass 1

- using the fix-point equations we get the following table for results after pass 1

block	in	out
$B1$	000 0000	111 0000
$B2$	111 0011	001 1110
$B3$	001 1110	000 1110
$B4$	001 1110	001 0111

Figure 3: Pass 1

Pass 2

- we reach a fix-point.

- the results are shown below:

block	*in*	*out*
$B1$	000 0000	111 0000
$B2$	111 1111	001 1110
$B3$	001 1110	000 1110
$B4$	001 1110	001 0111

Figure 4: Pass 2

Available expressions analysis

- an expression $x + y$ is *available* at a point p
  - if every path from the initial node to p evaluates $x + y$
  - after last such evaluation prior to reaching p there are no subsequent assignments to x or y.

Live-variable analysis

- a variable x is called *live* at a point p if x could be used along some path in the flow-graph starting at p.

- if the condition given above is not true, the variable x is called *dead*.