Data-flow Analysis - Part 3

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

Data-Flow Analysis Schema

- A *data-flow value* for a program point represents an abstraction of the set of all possible program states that can be observed for that point
- The set of all possible data-flow values is the *domain* for the application under consideration
  - Example: for the *reaching definitions* problem, the domain of data-flow values is the set of all subsets of of definitions in the program
  - A particular data-flow value is a set of definitions
- $IN[s]$ and $OUT[s]$: data-flow values *before* and *after* each statement s
- The *data-flow problem* is to find a solution to a set of constraints on $IN[s]$ and $OUT[s]$, for all statements s

Data-Flow Analysis Schema (2)

- Two kinds of constraints
  - Those based on the semantics of statements (*transfer functions*)
  - Those based on flow of control
- A DFA schema consists of
  - A control-flow graph
  - A direction of data-flow (forward or backward)
  - A set of data-flow values
  - A confluence operator (normally set union or intersection)
  - Transfer functions for each block
- We always compute *safe* estimates of data-flow values
- A decision or estimate is *safe* or *conservative*, if it never leads to a change in what the program computes (after the change)
- These safe values may be either subsets or supersets of actual values, based on the application

Live Variable Analysis

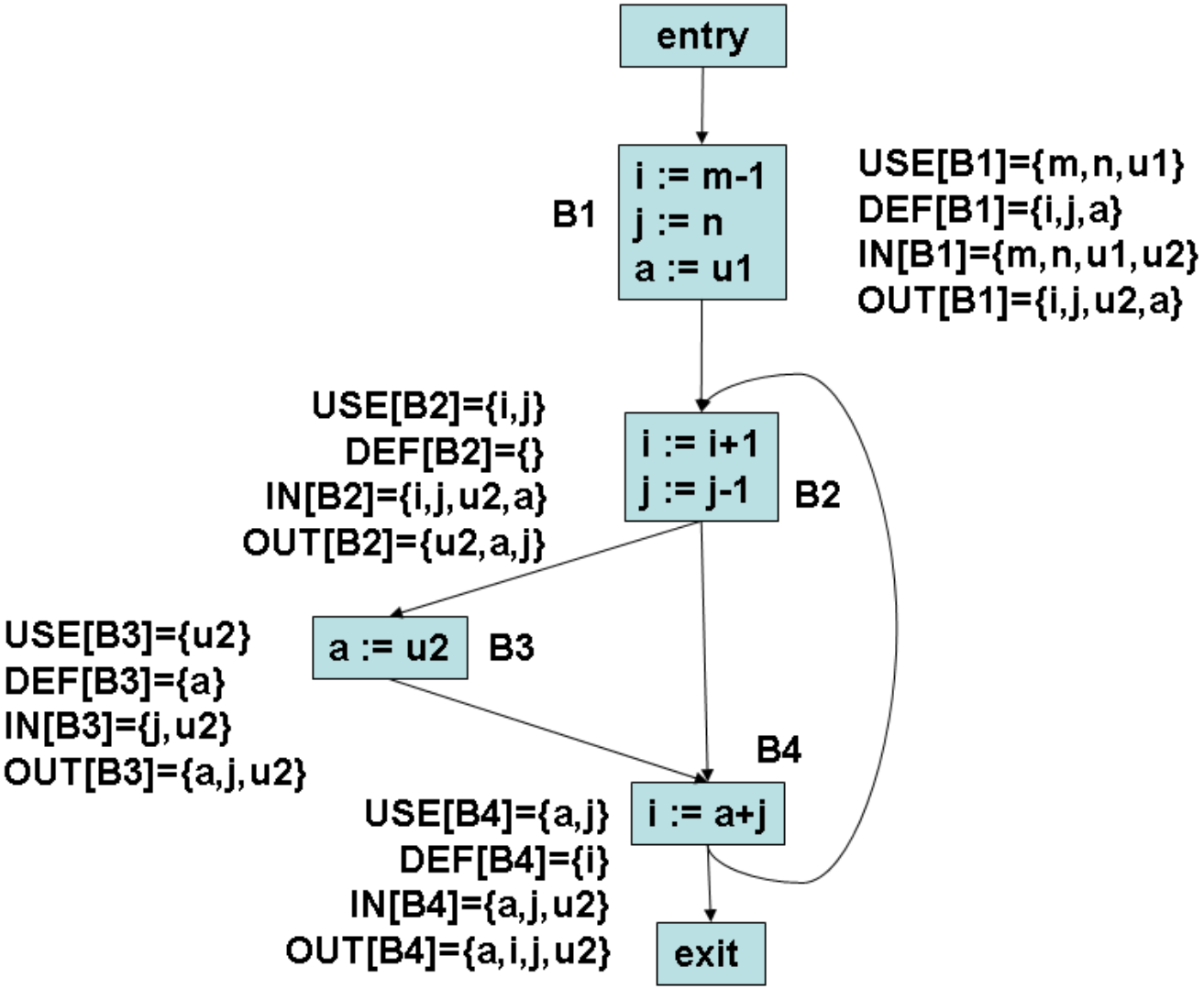
- The variable x is *live* at the point p, if the value of x at p could be used along some path in the flow graph, starting at p; otherwise, x is *dead* at p
- Sets of variables constitute the domain of data-flow values
- Backward flow problem, with confluence operator $\bigcup$
- $IN[B]$ is the set of variables live at the beginning of B
- $OUT[B]$ is the set of variables live just after B
- $DEF[B]$ is the set of variables definitely assigned values in B, prior to any use of that variable in B
- $USE[B]$ is the set of variables whose values may be used in B prior to any definition of the variable

$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

$$IN[B] = \phi, \text{for all } B \text{ (initialization only)}$$

Live Variable Analysis: An Example



Definition-Use Chains (d-u chains)

- For each definition, we wish to attach the statement numbers of the uses of that definition
- Such information is very useful in implementing register allocation, loop invariant code motion, etc.
- This problem can be transformed to the data-flow analysis problem of computing for a point p, the set of uses of a variable (say x), such that there is a path from p to the use of x, that does not redefine x.
- This information is represented as sets of (x, s) pairs, where x is the variable used in statement s
- In live variable analysis, we need information on whether a variable is used later, but in (x, s) computation, we also need the statment numbers of the uses
- The data-flow equations are similar to that of LV analysis
- Once $IN[B]$ and $OUT[B]$ are computed, d-u chains can be computed using a method similar to that of u-d chains

Data-flow Analysis for (x,s) pairs

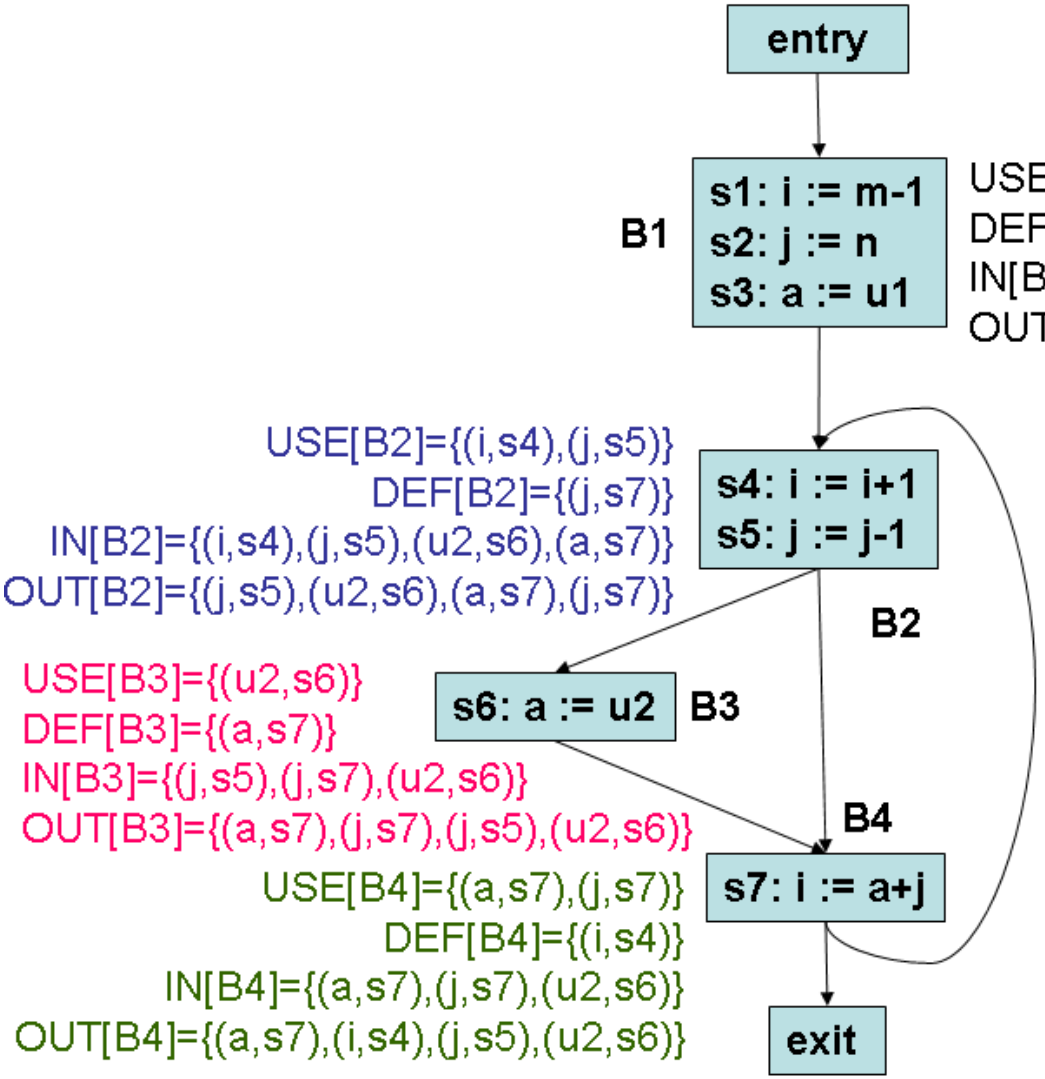
- Sets of pairs (x,s) constitute the domain of data-flow values
- Backward flow problem, with confluence operator $\bigcup$
- $USE[B]$ is the set of pairs (x, s), such that s is a statement in B which uses variable x and such that no prior definition of x occurs in B
- $DEF[B]$ is the set of pairs (x, s), such that s is a statement which uses x, s is *not in* B, and B contains a definition of x
- $IN[B]$ ($OUT[B]$, resp.) is the set of pairs (x, s), such that statement s uses variable x and the value of x at $IN[B]$ ($OUT[B]$, resp.) has not been modified along the path from $IN[B]$ ($OUT[B]$, resp.) to s

$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

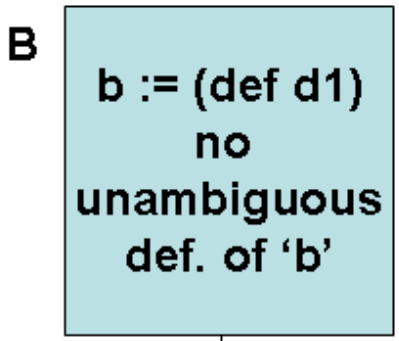
$$IN[B] = \phi, \text{for all } B \text{ (initialization only)}$$

Definition-Use Chain Example

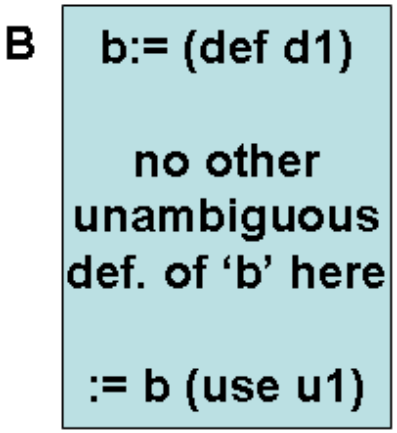


def	d-u chain
(i,s1)	(s4)
(j,s2)	(s5)
(a,s3)	(s7)
(i,s4)	()
(j,s5)	(s5,s7)
(a,s6)	(s7)
(i,s7)	(s4)

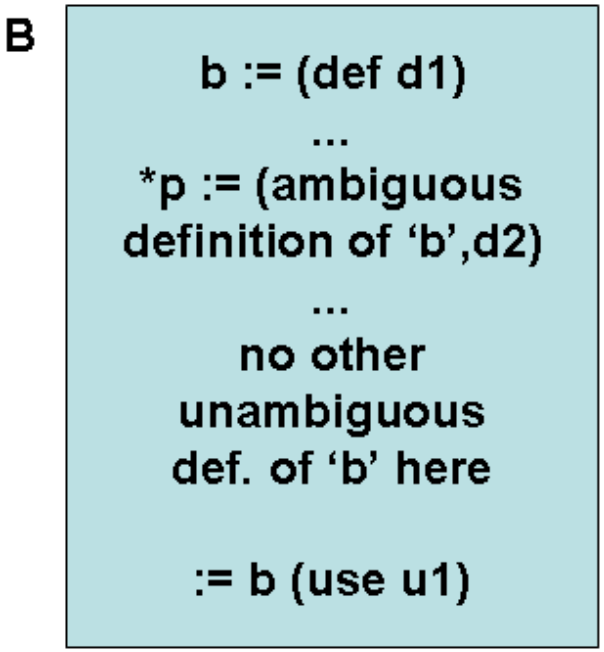
Definition-Use Chain Construction



attach to du-chain of d1, stmts s_i of all use pairs (b,s_i) in OUT[B]



attach use u1 to du-chain of def d1

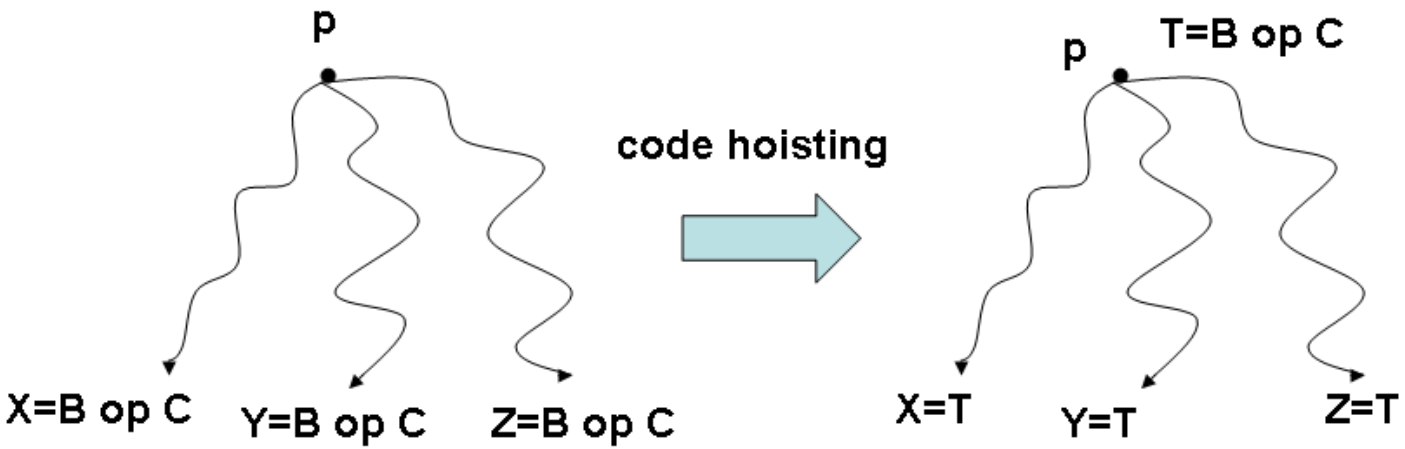


attach use u1 to du-chains of both def d1 and def d2

Three cases while constructing d-u chains from the (x,s) pairs

Very Busy Expressions *or* Anticipated Expressions

- An expression *B op C* is very busy or anticipated at a point *p*, if along every path from *p*, we come to a computation of *B op C* before any computation of *B* or *C*
- Useful in code hoisting and partial redundancy elimination
- Code hoisting does not reduce time, but reduces space
- We must make sure that no use of *B op C* (from X,Y, or Z below) has any definition of *B* or *C* reaching it without passing through *p*

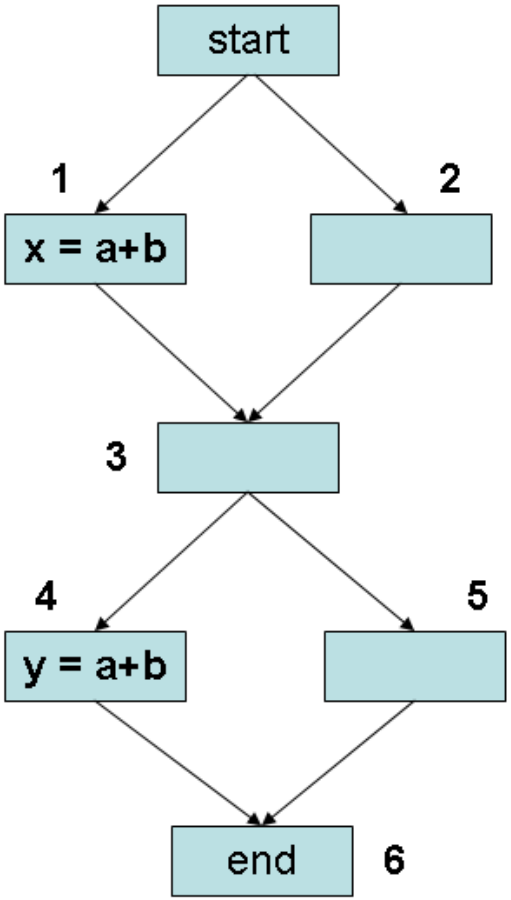


Very Busy Expressions *or* Anticipated Expressions (2)

- Sets of expressions constitute the domain of data-flow values
- Backward flow analysis with $\bigcap$ as confluence operator
- $V\_USE[n]$ is the set of expressions $B\ op\ C$ computed in n with no prior definition of B or C in n
- $V\_DEF[n]$ is the set of expressions $B\ op\ C$ in U (the universal set of expressions) for which either B or C is defined in n, prior to any computation of $B\ op\ C$

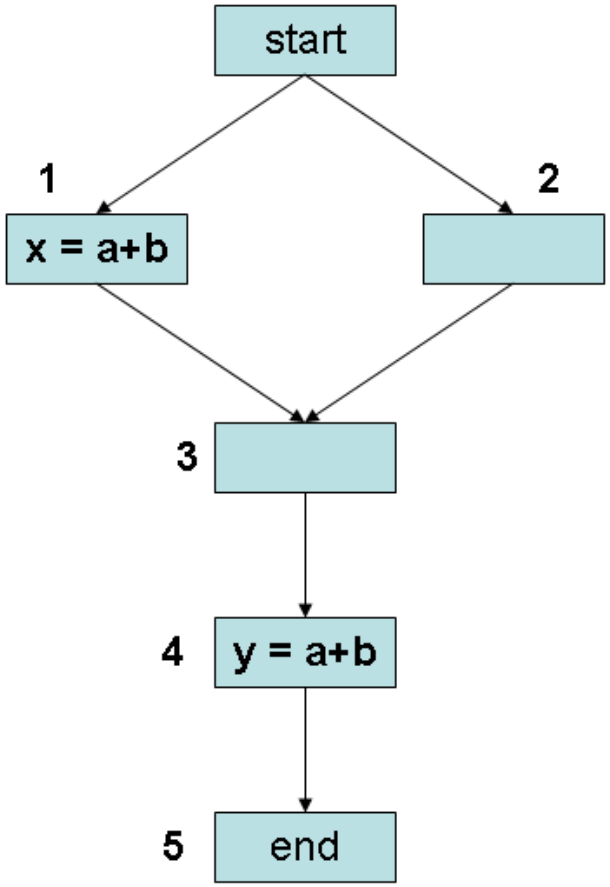
$$
\begin{aligned}
OUT[n] &= \bigcap_{S\ is\ a\ successor\ of\ n} IN[S] \\
IN[n] &= V\_USE[n] \bigcup (OUT[n] - V\_DEF[n]) \\
IN[n] &= U, for\ all\ n\ (initialization\ only)
\end{aligned}
$$

Anticipated Expressions - An Example



(a)

a+b is anticipated at: entry to 1 and 4
a+b is not anticipated at: all other points



(b)

a+b is anticipated at all points,
except at exit of 4 and entry of 5

Data-Flow Problems: A Summary - 1

The Reaching Definitions Problem

- Domain of data-flow values: sets of definitions
- Direction: Forwards
- Confluence operator: $\cup$
- Initialization: $IN[B] = \phi$
- Equations:

$$IN[B] = \bigcup_{P \text{ is a predecessor of } B} OUT[P]$$

$$OUT[B] = GEN[B] \bigcup (IN[B] - KILL[B])$$

Data-Flow Problems: A Summary - 2

The Available Expressions Problem

- Domain of data-flow values: sets of expressions
- Direction: Forwards
- Confluence operator: $\cap$
- Initialization: $IN[B] = U$
- Equations:

$$
\begin{aligned}
IN[B] &= \bigcap_{P \text{ is a predecessor of } B} OUT[P] \\
OUT[B] &= e\_gen[B] \bigcup (IN[B] - e\_kill[B]) \\
IN[B1] &= \phi
\end{aligned}
$$

Data-Flow Problems: A Summary - 3

The Live Variable Analysis Problem

- Domain of data-flow values: sets of variables
- Direction: backwards
- Confluence operator: $\cup$
- Initialization: $IN[B] = \phi$
- Equations:

$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Data-Flow Problems: A Summary - 4

The Anticipated Expressions (Very Busy Expressions) Problem

- Domain of data-flow values: sets of expressions
- Direction: backwards
- Confluence operator: $\cap$
- Initialization: $IN[B] = U$
- Equations:

$$OUT[B] = \bigcap_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = V\_USE[B] \bigcup (OUT[B] - V\_DEF[B])$$