GLOBAL DATA FLOW ANALYSIS

Data-flow information can be collected by setting up and solving systems of equations that relate information at various points in a program. A typical equation has the form

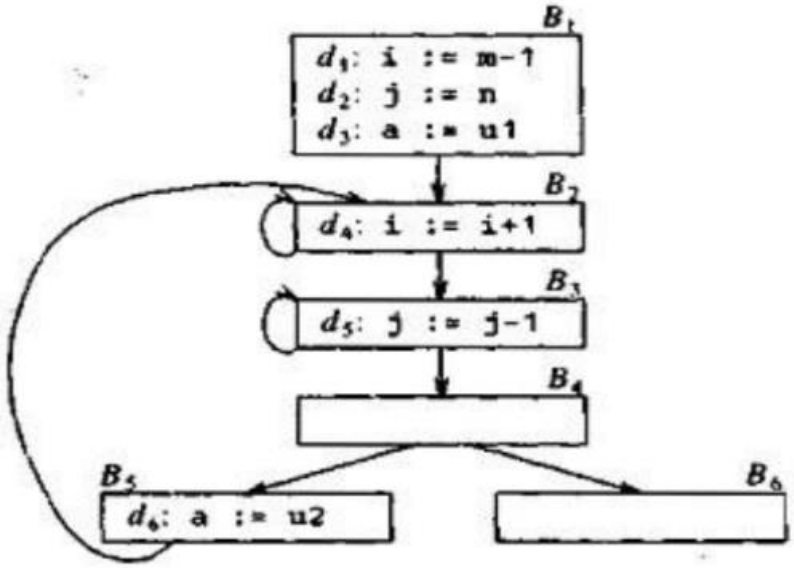
out[S] = gen[S] U (in[S] - kill[S])

The information at the end of a statement is either generated within the statement, or enters at the beginning and is not killed as control flows through the statement. Such equations are called data-flow equations. The details of how data-flow equations are set up and solved depend on three factors.

1. The notions of generating and killing depend on the desired information. i.e., on the data-flow analysis problem to be solved. Moreover, for some problems, instead of proceeding along with the flow of control and defining out[S] in terms of in[S], we need to proceed backwards and define in[S] in terms of out[S].
2. Since data flows along control paths, data-flow analysis is affected by the control constructs in a program. In fact, when we write out[S] we implicitly assume that there is unique end point where control leaves the statement.
3. There are subtleties that go along with such statements as procedure calls, assignments through pointer variables, and even assignments to array variables.

Points and Paths

Within a basic block, we talk of the point between two adjacent statements, as well as the point before the first statement and after the last. Thus, block B1 has four points: one before any of the assignments and one after each of the three assignments.



Now, let us take a global view and consider all the points in all the blocks. A path from P1 to Pn is a sequence of points P1, P2, . . . , Pn such that for each I between 1 and n-1, either

1. Pi is the point immediately preceding a statement and Pi+1 is the point immediately following that statement in the same block, or
2. Pi is the end of some block and Pi+1 is the beginning of a successor block.

Reaching Definitions

A definition of a variable x is a statement that assigns, or may assign, a value to x. The most common forms of definition are assignments to x and statements that read a value from an I/O device and store it in x. These statements certainly define a value for x, and they are referred to as unambiguous definitions of x.

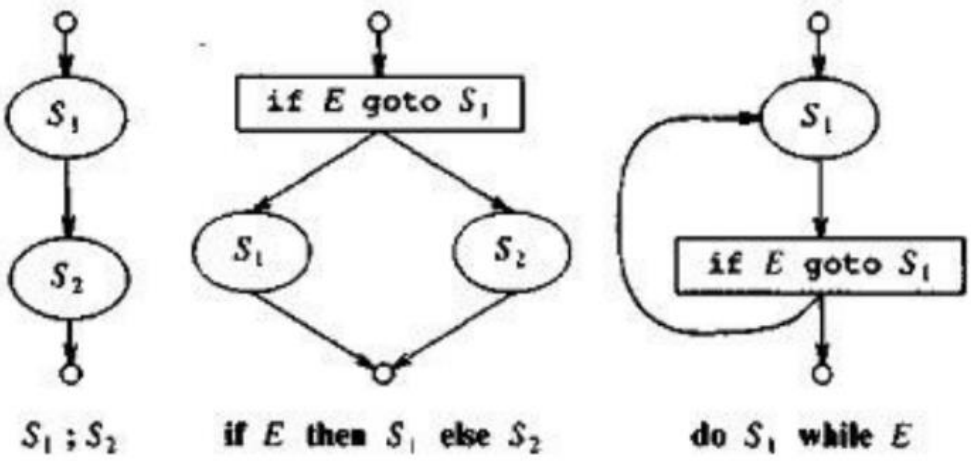
There are certain other kinds of statements that may define a value for x; they are called ambiguous definitions. The most usual forms of ambiguous definitions or x are:

1. A call of a procedure with x as a parameter or a procedure that can access x because x is in the scope of the procedure. We also have to consider the possibility of "aliasing;" where x is not in the scope of the procedure, but x has been identified with another variable that is passed as a parameter or is in the scope.
2. An assignment through a pointer that could refer to x. For example, the assignment *q:=y is a definition of x if it is possible that q points to x.

Data-Flow Analysis of Structured Programs

Consider the production

S -> id :=E | S;S | if E then S else S | do S while E E -> id + id | id



We define a portion or a flow graph called a region to be a set of nodes N that includes a header, which dominates all other nodes in the region. All edges between nodes in N are in the region, except for some that enter the header.

Representation of Sets

Sets of definitions, such as gen[S] and kill[S], can be represented compactly using bit vectors. We assign a number to each definition of interest in the flow graph. Then the bit vector representing a set of definitions will have 1 in positions i if and only if the definition numbered i is in the set.

A bit-vector representation for sets also allows set operations to be implemented efficiently. The union and intersection of two sets can be implemented by logical or and logical and, respectively, basic operations in most systems-oriented programming languages. The difference A-B of sets A and B can be implemented by taking the complement of B and then using logical and to compute A^ ¬ B.