Program Dependencies

Reading assignment

- M. C. Thompson, D. J. Richardson and L. A. Clarke, "An Information Flow Model of Fault Detection", Proceedings of the International Symposium on Software Testing and Analysis, (ISSTA), Boston, MA, June 1993, pp.182-192

Today's reading

- A. Podgurski and L. A. Clarke, "A Formal Model of Program Dependencies and Its Implications for Software Testing, Debugging, and Maintenance", IEEE Transactions on Software Engineering, 16 (9), September 1990, pp. 965-979.
- Background
  - M. Weiser, "Program Slicing," Proceedings of the Fifth International Conference on Software Engineering, San Diego, Ca 1981, pp. 439-449.
  - D. W. Binkley and K. B. Gallagher, "Program Slicing," Advances in Computers, Vol. 43, M. Zelkowitz, editor, Academic Press, 1996 pp. 1-50.

Program Slice

- Introduced by Mark Weiser in 1979
- Argued it was a mental abstraction that programmers used when debugging
- Program slice S is a reduced, executable program obtained from program P by removing statements from P, such that S replicates part of the behavior of P

Program Slice

```
read n;
i := 1;
sum := 0;
product := 1;
while i ≤ n do
        sum := sum +1;
        product := product * i;
        i:= i+1;
endwhile;
write sum;
write product;
```

```
read n;
i := 1;

product := 1;
while i ≤ n do

        product := product * i;
        i:= i+1;
endwhile;


write product;
```

Original Slicing Concept

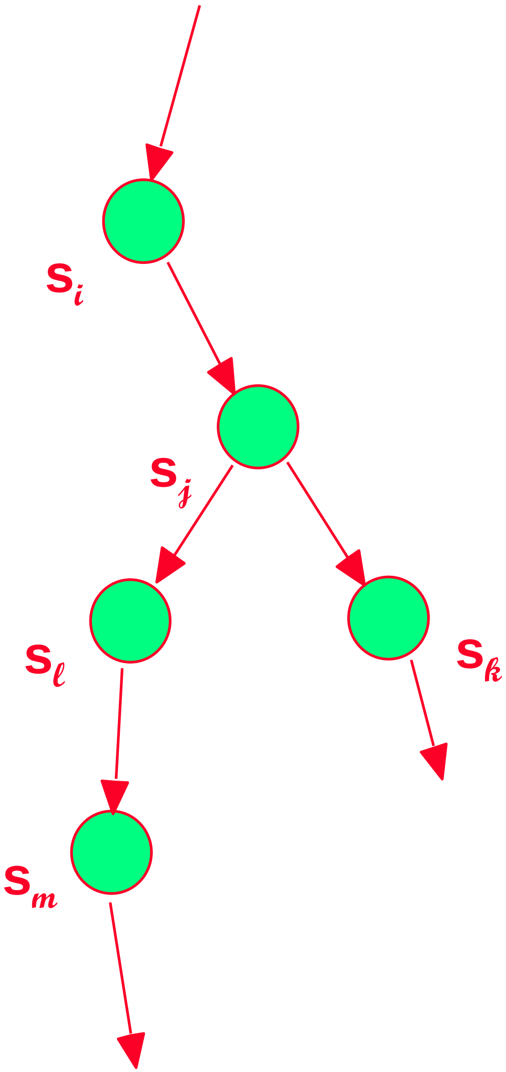
- Based on statements
- Algorithm restricted to structured programs
- Missed some relationships
- Foundation for considerable work
  - Podgurski and Clarke generalized some of the concepts
    - Language-independent model of dependence
    - More general model of control flow
      - Weak and strong control flow

Applications of Program Slicing/Dependence

- Debugging
- Data/control flow testing criteria
- Software understanding
- Maintenance

Program Dependencies

- s_k is semantically dependent on s_i if the semantics of s_i can affect the execution behavior of s_k
- In general, can't determine semantic dependence



Syntatic Dependence

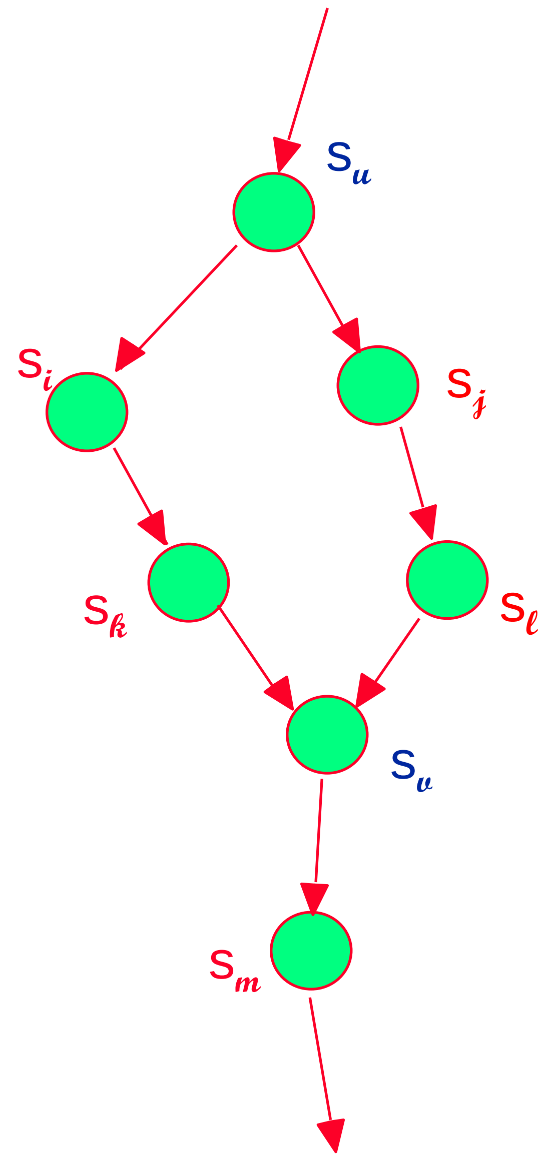
- Can we find syntactic dependence relations that "approximate" semantic dependence?
  - that define necessary conditions for semantic dependence
  - that are defined in terms of a language-independent, graph-theoretic model that can be efficiently computed

Forward Dominators

- let G(N,E) be a control flow graph, where s_u, s_v, and s_f are nodes in G and s_f is the final node
  - a node s_v forward dominates s_u iff every s_u ->s_f path in G contains s_v
  - s_v properly forward dominates s_u iff
    - $s_u \neq s_v$ and s_v forward dominates s_u

Immediate Forward Dominators

A node s_v is the the immediate forward dominator of s_u, $s_u \neq s_\ell$ if it is the node that is the first proper forward dominator of s_v to occur on every s_u->s_ℓ path in G



after a branch, this is the point where all paths come together

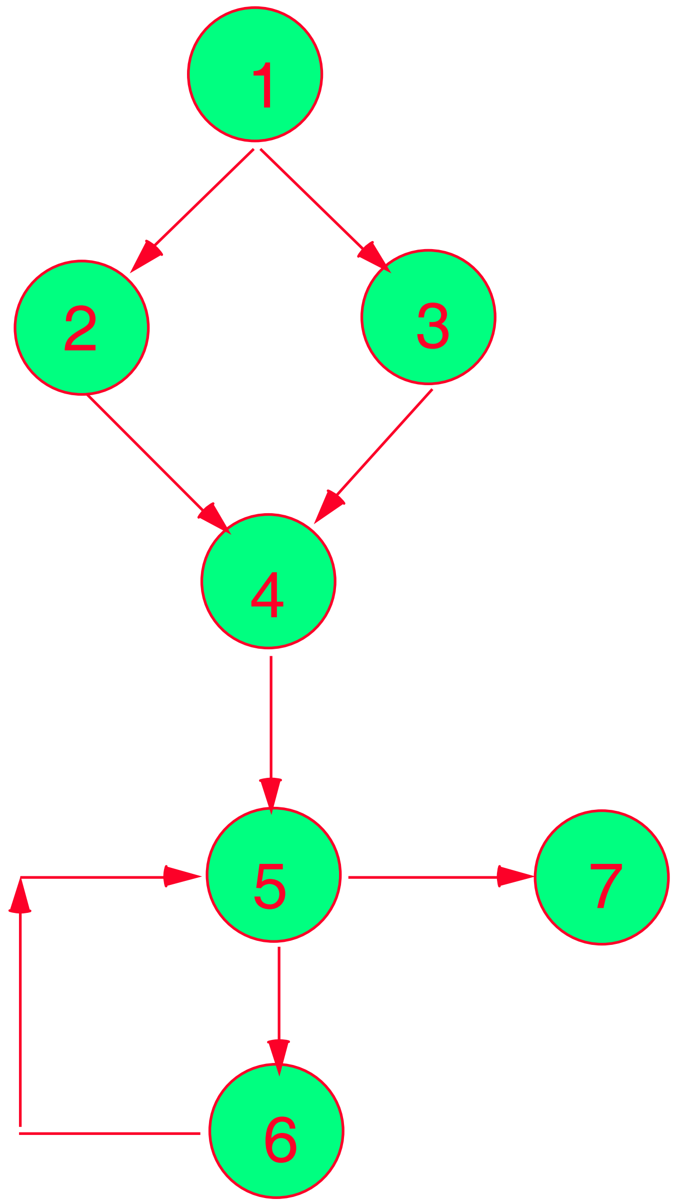
Example

7 forward dominates all nodes

ifd(5) = 7

ifd(1)= 4

ifd(4)= 5

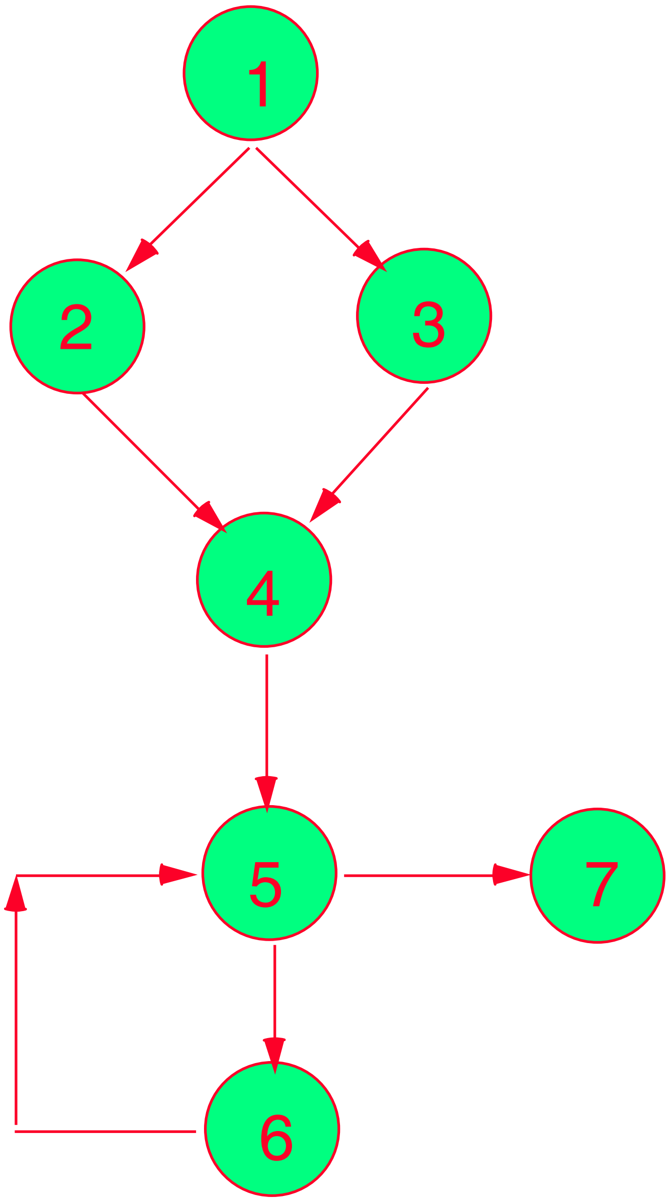


Control Dependence

- s_v is control dependent on s_u iff there exists a path $s_u \bullet P \bullet s_v$ not containing the immediate forward dominator of s_u
  - bodies of structured constructs are control dependent on the start of the construct

Example

- 2 and 3 are control dependent on 1
- 6 is control dependent on 5



Direct Data Dependence

Assume G(N,E) is a control flow graph, where Def(s_n) are the variables defined at node s_n and Use(s_n) are the variables referenced at node s_n.

if path P = $s_{i1}, \ldots, s_{in}$ then Def(P)= $\bigcup_j$ Def(s_{ij})

node s_v is directly data dependent on node s_u iff there is a path $s_u \bullet P \bullet s_v$ such that

$$(Def(s_u) \cap Use(s_v)) - Def(P) \neq 0$$

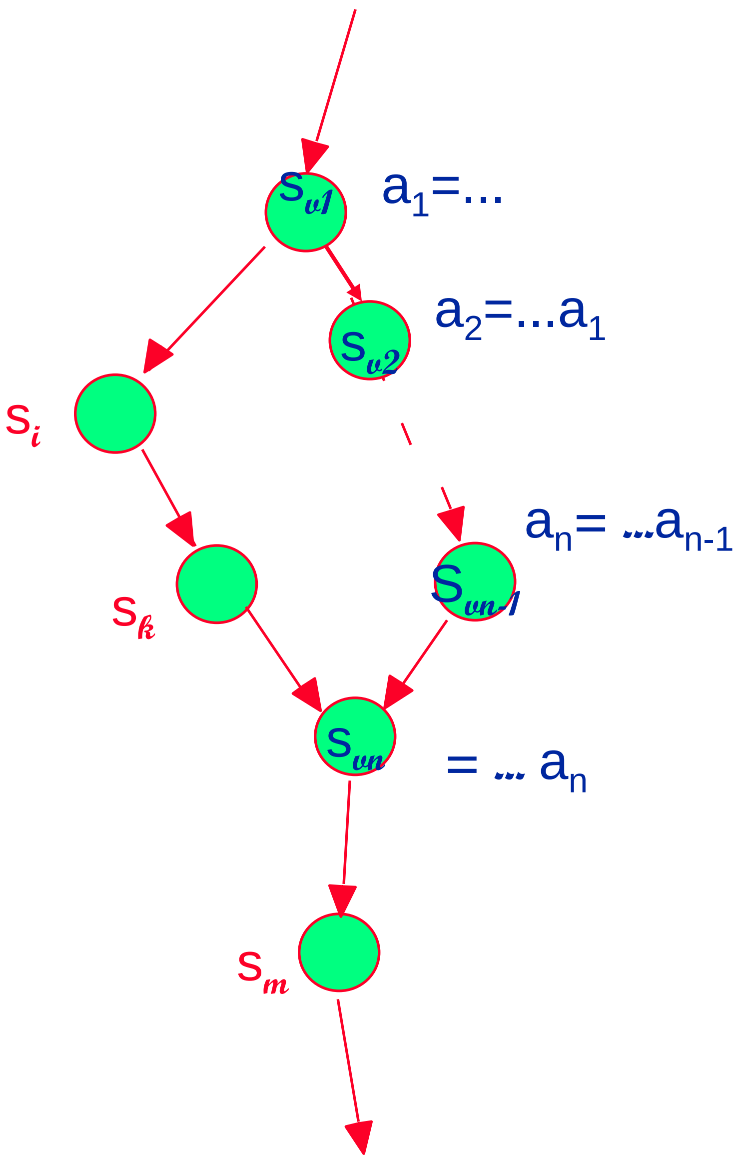
In other words

$$(Def(s_u) \cap Use(s_v)) - Def(P) \neq 0$$

This is just a set theoretic way of saying that there is at least one variable, say x, defined at node s_u *that is used at node* s_v *and there is a def-clear path with respect to x from* s_u *to* s_v

Data Dependence

- node s_v is **data dependent** on s_u iff there is a path $s_{v1},...,s_{vn}$ such that u=v1, v=vn, and S_{vi} is directly data dependent on S_{vi+1} for all i, $1 \leq i < n$

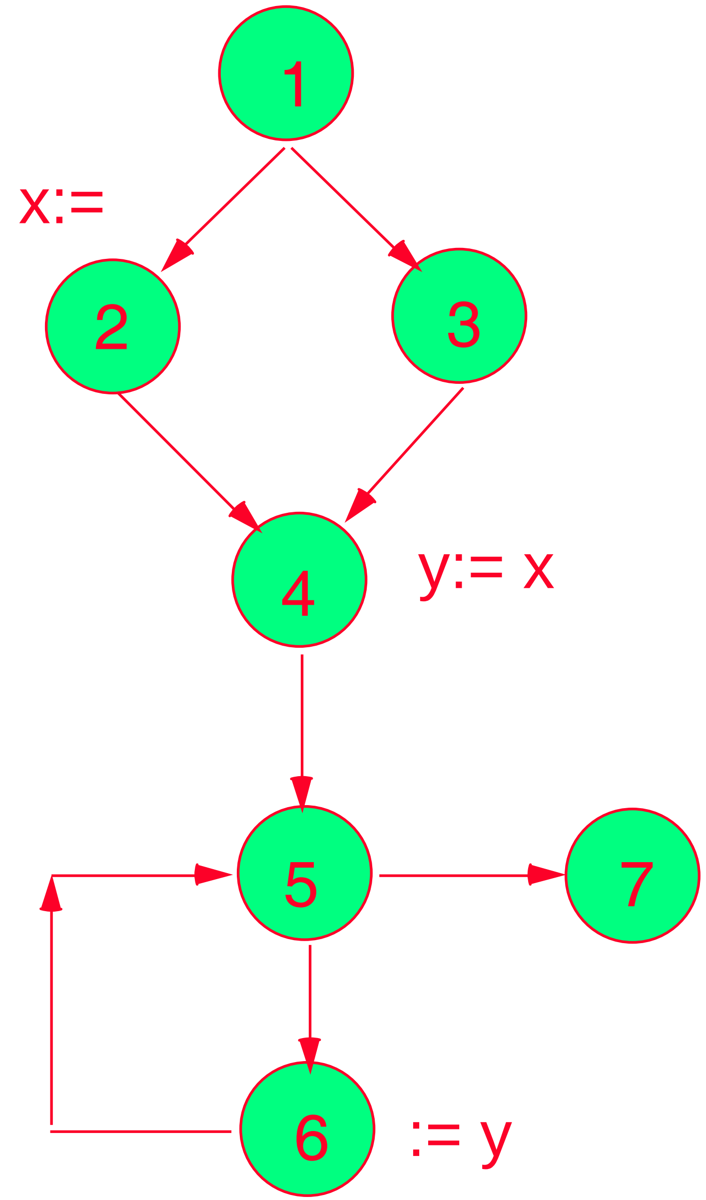


Example

- 4 is directly data dep. on 2
- 6 is directly data dep. on 4
- 6 is data dependent on 2

directly data dependent is the same as the def-use relationship defined by Rapps and Weyuker

data dependent is the same as the chains of def ref used by Ntafos, but without a bound



Syntactic Dependence

- node S_v is **syntactically dependent** on S_u iff there is a path $S_{v1},...,S_{vn}$ of nodes such that u=v1, v=vn, and S_{vi+1} is **data or control** dependent on S_{vi} for all i, $1 \leq i < n$
  - combines data dependence and control dependence
  - sometimes called **information flow**
  - syntactic dependence over-approximates semantic dependence
    - **Why?**

Syntactic Dependence

Control flow

(1,2)

(1,3)

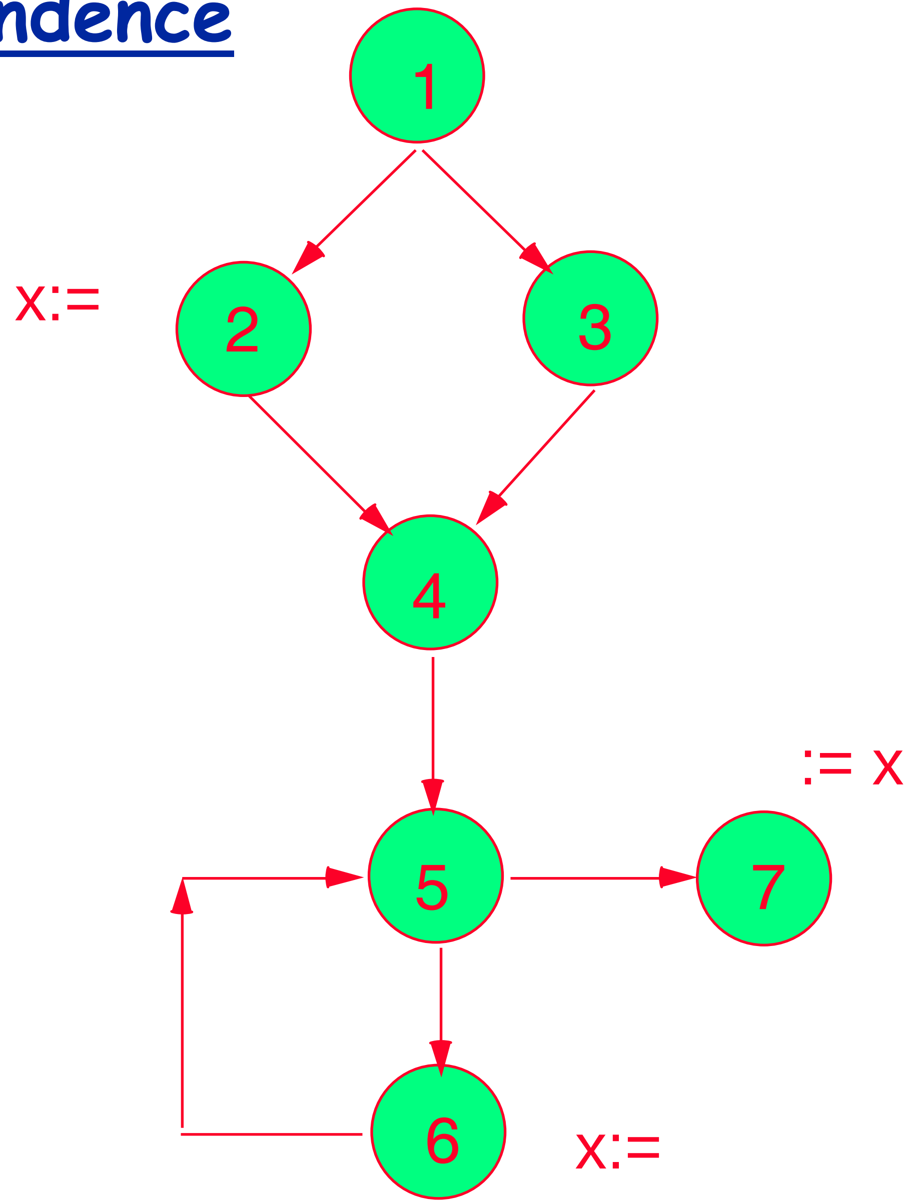
(5,7)

(5,6)

Direct Data flow

(2,7)

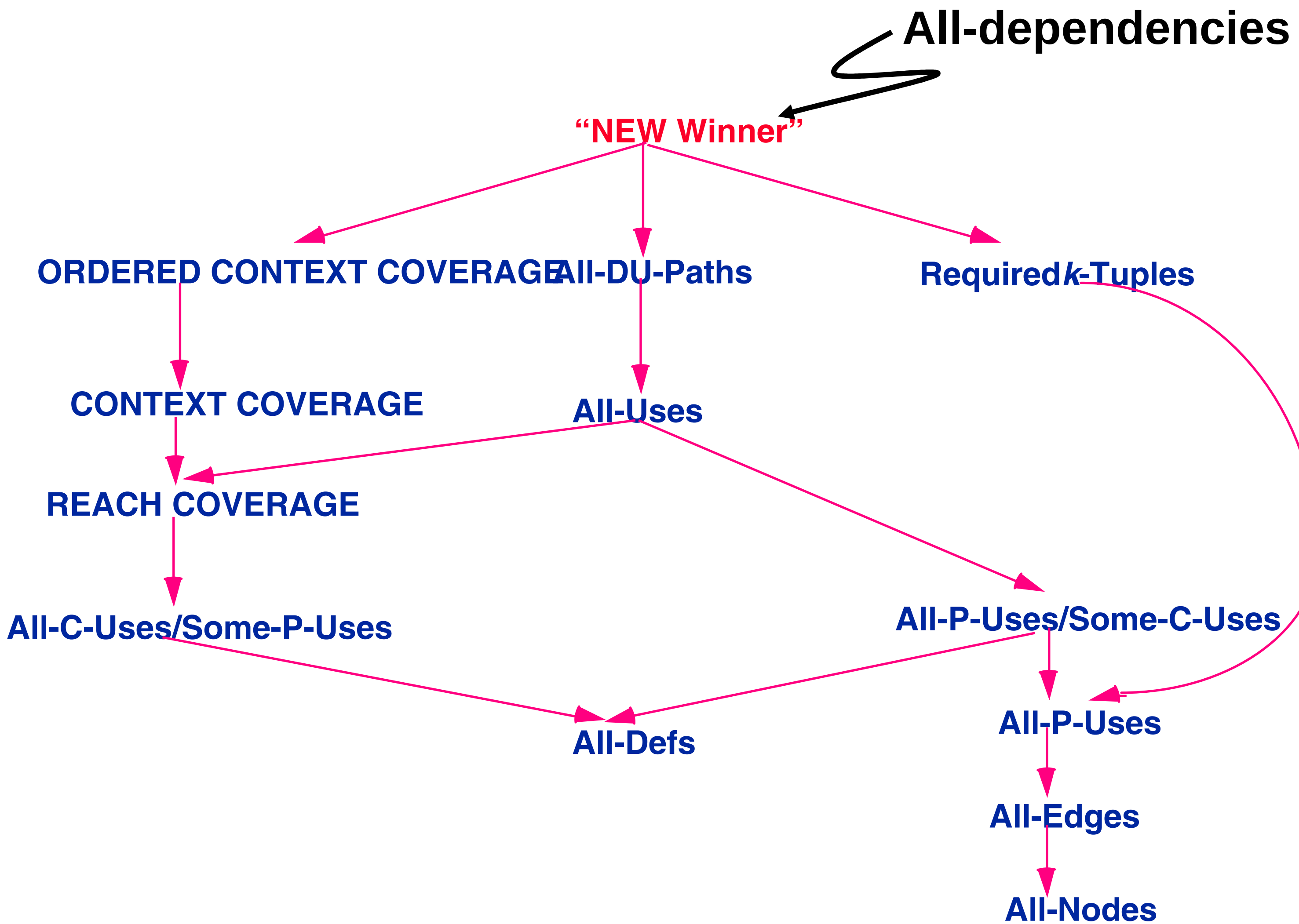
(6,7)



Data (and control) flow coverage criteria

- coverage criteria exercise subsets of control and data dependencies in the hope of exposing faults
- Rapps and Weyuker, Ntafos, Laski and Korel selected different subsets of information flow
- need experimental data to know which are the most effective subsets
  - intuitively, direct data dependence and control dependence are appealing
  - relatively easy to achieve at least 85% coverage with automated support

Data Flow/Control Flow Coverage revisited

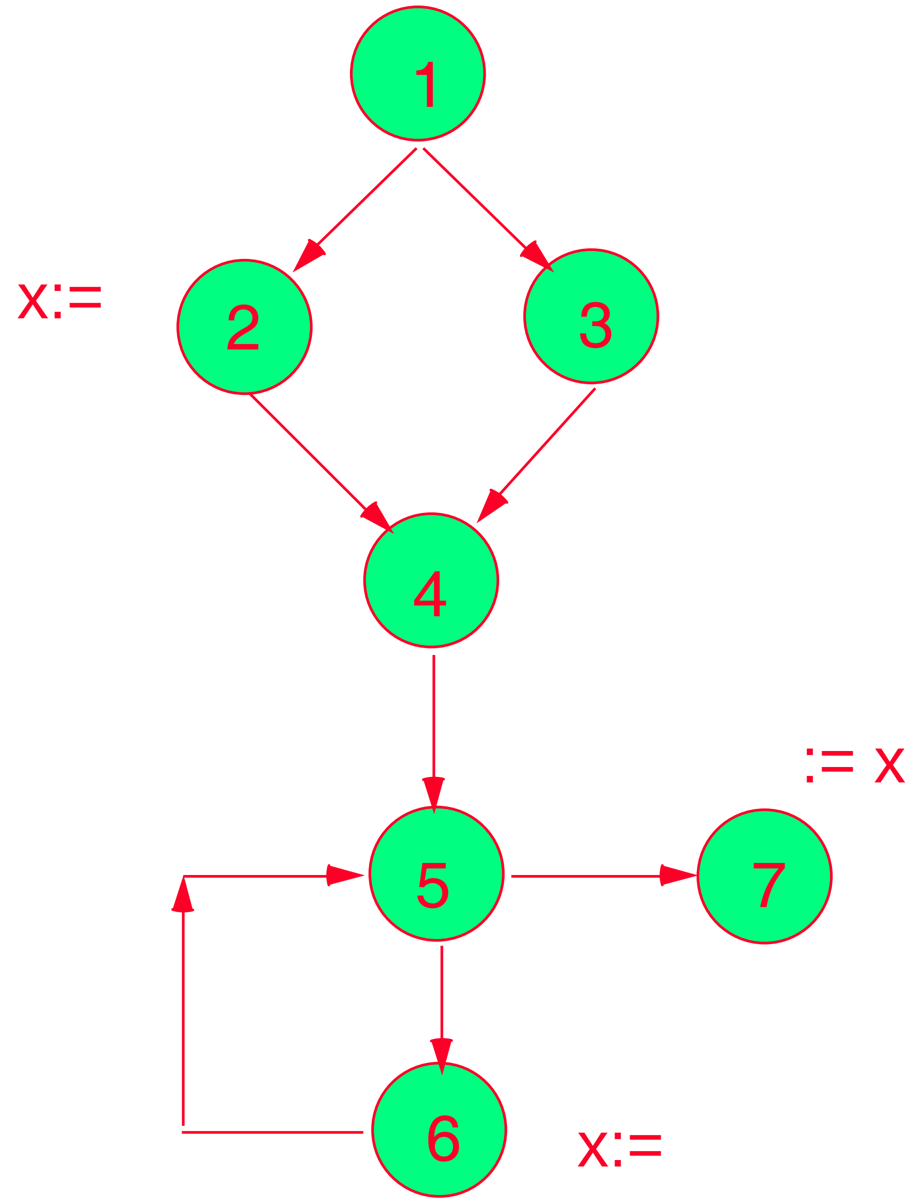


Applications

- Debugging
- Data/control flow testing criteria
- Software understanding
- Maintenance

Symmetric Relationship

- dep(s_i, s_j) is true if s_j is syntactically dependent on s_i
- dep(?, s_j) = {s_i | dep(s_i,s_j)} is the set of nodes that can syntactically affect s_j
- dep(s_i, ?) ={s_j | dep(s_i,s_j)} is the set of nodes that can be syntactically affected by s_i



Debugging Dependencies

- Which statements could have caused an observed failure?

- If s_v computes an erroneous value, want to know the statements s_v is dependent upon?

    - dep(?, S_v)

Maintenance Dependencies

- Which statements will be affected by a change?
  ->which statements are dependent upon s_u
  dep(s_u,?)
- Will a particular statement be affected by a change?
  - Is there a dependency between S_u and S_v?
    dep(S_u,S_v)?
- Which statements could affect "this" statement?
  - Which statements are statement S_v dependent on?
    dep(?,S_v)

Kinds of flow

- Forward flow
  - dep(S_u,?)

- Backward flow
  - dep(?,S_v)

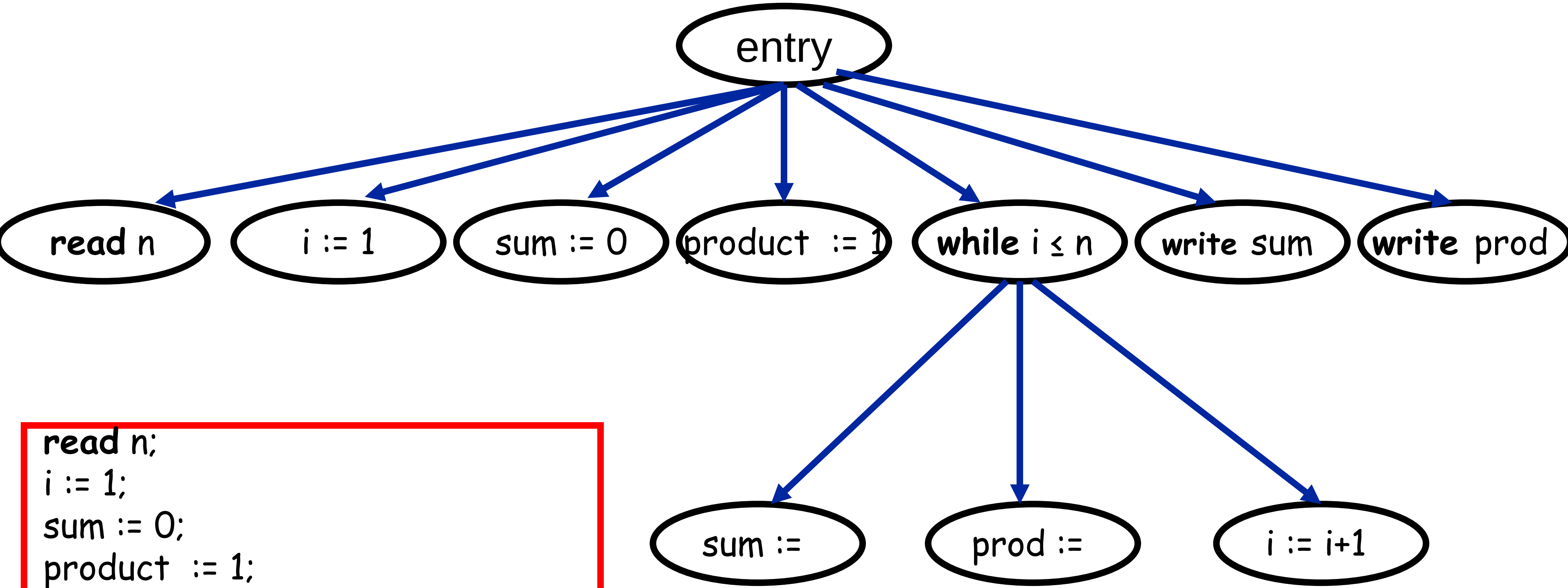
Program Dependence Graph

- Originally proposed by Ottenstein and Ottenstein(Ott), 1984
- Nodes correspond to statements
- Edges correspond to data or control dependencies
- A slice corresponds to all nodes that are reachable from a selected node (forward slice)

Program dependence graph example

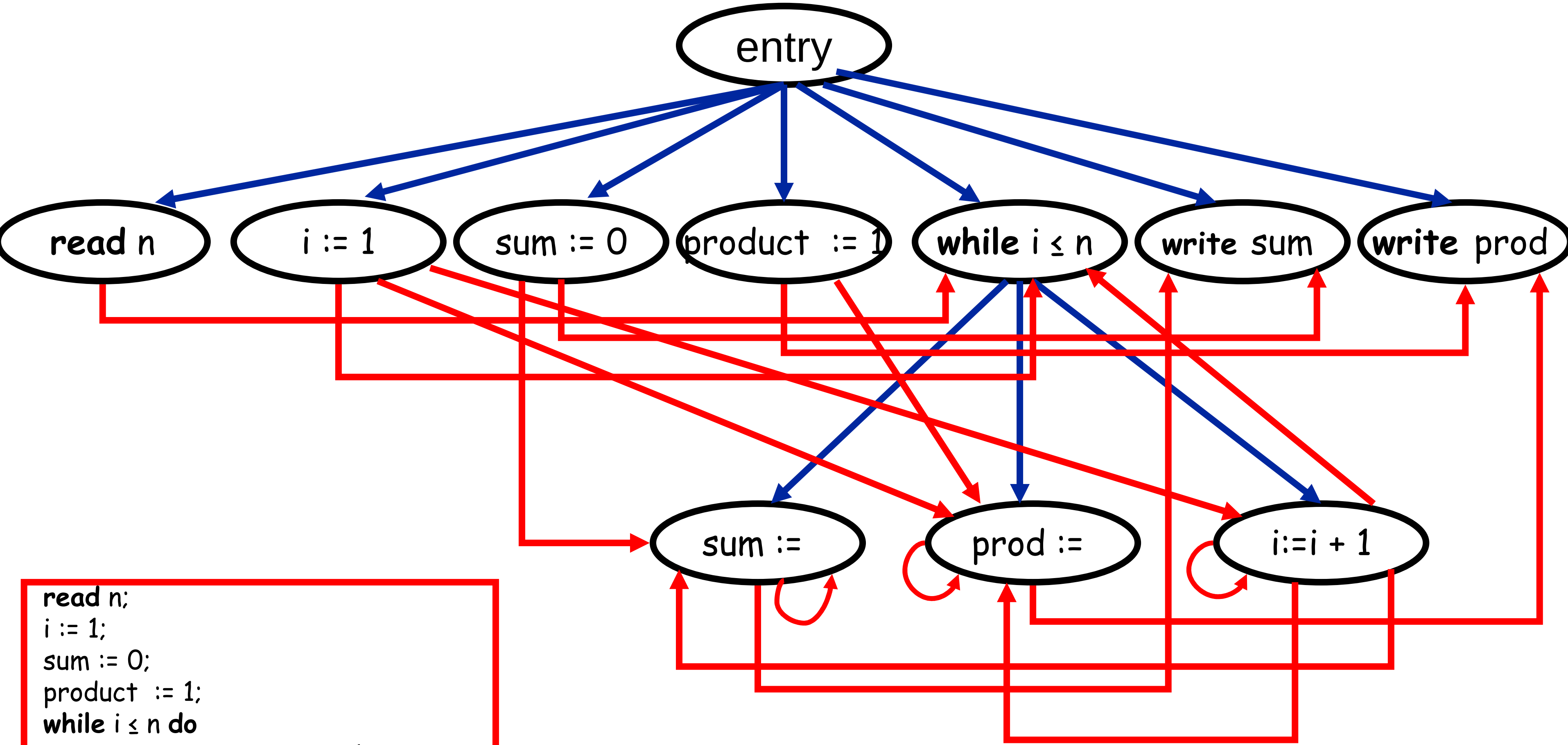
```
read n;
i := 1;
sum := 0;
product  := 1;
while i ≤ n do
        sum := sum +1;
        product := product * i;
        i:= i+1;
endwhile;
write sum;
write product;
```

Program Dependence Graph Example



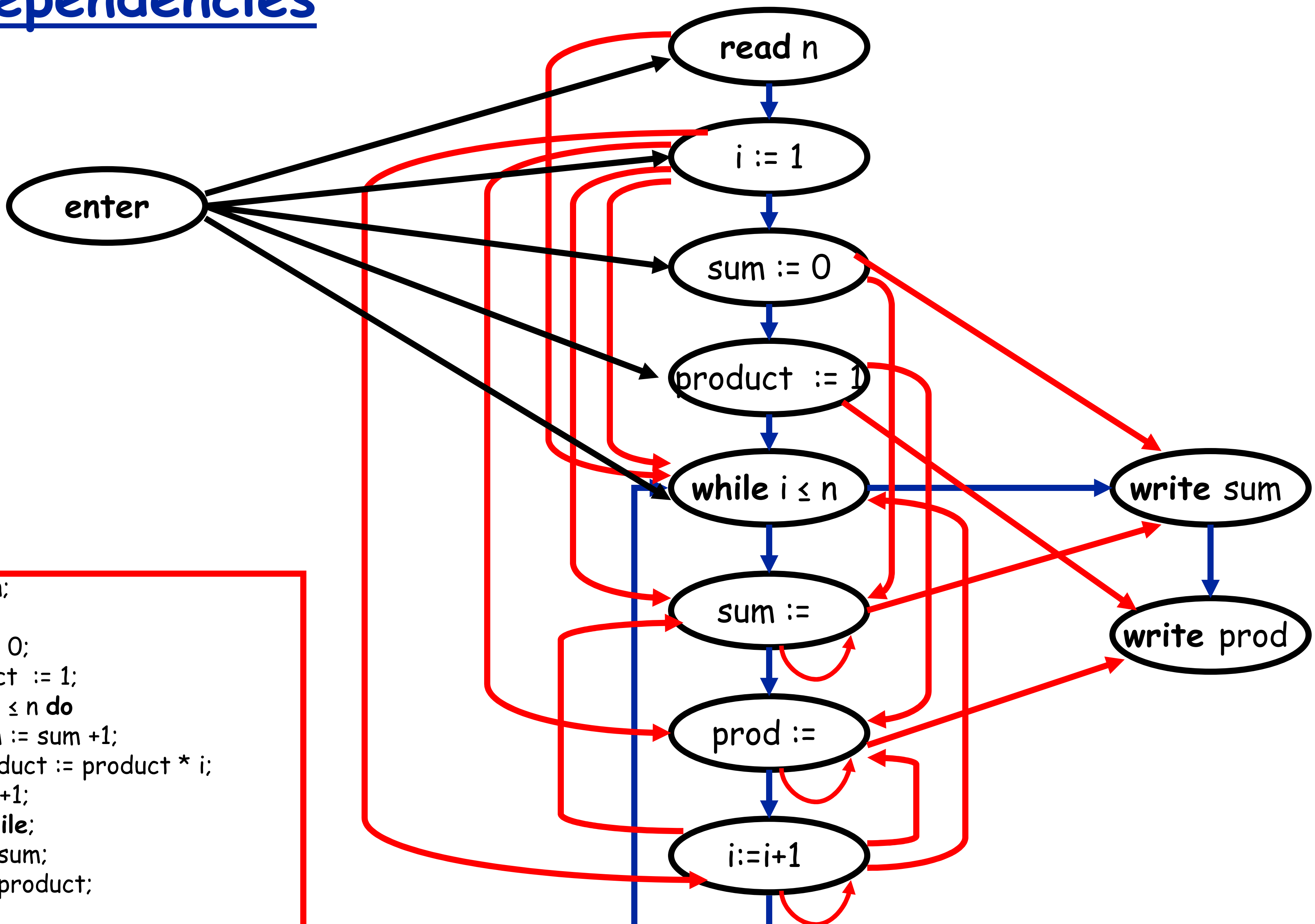

```
read n;
i := 1;
sum := 0;
product  := 1;
while i ≤ n do
    sum := sum +1;
    product := product * i;
    i:= i+1;
endwhile;
write sum;
write product;
```

Program Dependence Graph Example




```
read n;
i := 1;
sum := 0;
product  := 1;
while i ≤ n do
        sum := sum +1;
        product := product * i;
        i:= i+1;
endwhile;
write sum;
write product;
```

Control Flow Graph Model w/ Data Dependencies




```
read n;
i := 1;
sum := 0;
product := 1;
while i ≤ n do
    sum := sum +1;
    product := product * i;
    i:= i+1;
endwhile;
write sum;
write product;
```

Problems with dependence analysis/slicing

- In practice, a program slice is often too big to be useful
- Infeasible paths lead to imprecision
- Complex data structures lead to imprecision

```
A[i]:=
    …
B[j] := A[k]
```

- Need to use an efficient, interprocedural algorithm

Refining program dependencies

- Dependence/slice wrt a criteria
- Dynamic dependence/slice

Levels of Granularity

- by statement
- by entity
  - e.g., x:= y + z
  - only look at dependencies on z
  - only look at data dependencies on z
  - only look at direct data dependencies on z
- by component
  - e.g., TASC avionics maintenance system

Dynamic Slice

- First proposed by Laski and Korel, 1988
- Only provides those dependencies that were exercised during a particular execution
- Could also be further refined according to some criteria
  - E.g., dynamic slice and depends on statement n

Conclusion

- program dependencies provide a theory for restricting/focusing attention
  - can allow users to select and refine focus of attention
  - can support different levels of granularity
- Can be used for software understanding, regression testing, debugging, maintenance, and data/control test coverage criteria

Conclusion

- for selecting test cases
  - syntactic dependence alone is not adequate
    - the number of syntactic dependencies in a program can be quadratic in the number of statements
    - a given syntactic dependence may be demonstrated by (infinitely) many paths
    - propagation of a fault through a particular path may depend on the selection of input data

      ⇒must use semantic information