## Class 7

- Review; questions
- Discuss Problem Set 3
- Assign (see Schedule for links)
  - Horwitz, Reps, Binkley SDG paper
  - Problem Set 4:  due 9/15/09



## Program Slicing (cont'd)

1. Slicing overview
2. Types of slices, levels of slices
   - Backward
   - Forward
   - Execution
   - Executable
   - Dynamic
3. Methods for computing slices
   - Dataflow on CFG
   - Graph reachability on the PDG

## Methods (Data Flow on the CFG)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

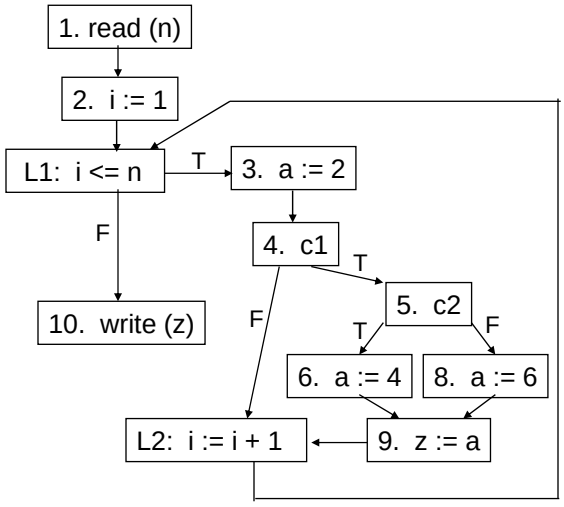
Criterion <10, sum>

## Methods (Data Flow on the CFG)

- Example on board

## Methods (Data Flow on the CFG)

```
1.  read (n)
2.  for I := 1 to n do
3.     a := 2
4.     if c1 then
5.        if c2 then
6.           a := 4
7.        else
8.           a := 6
9.        z := a
10. write (z)
```




## Methods (Data Flow on the CFG)

Compute slice for criterion <10, z>

- Create sets Rel(n) for all nodes n and Slice
- Rel(10) is {z}; Slice={10}; put preds of 10 on Worklist
- Remove m from Worklist; compute Rel(m); put m in Slice if required; if m put in Slice, put node(s) on which it is control dependent in Slice; put preds on Worklist if Rel(m) is not empty; until Worklist is empty
- Worked through algorithm in class

## Methods (Data Flow on the CFG)

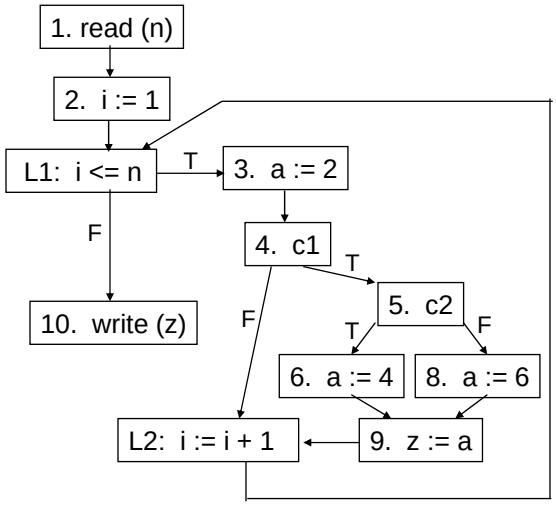
- Issues with algorithm
  - Interprocedural Slicing (more on Thursday)
    - Solution proposed by Weiser
    - Can go up or down procedure calls
    - Actual parameters of function call are changed to call parameters (or the opposite)
    - Variables not in scope are removed
    - Is not “context sensitive” – too inaccurate

## Methods (Graph Reachability on PDG)

```
1.  read (n)
2.  for I := 1 to n do
3.     a := 2
4.     if c1 then
5.        if c2 then
6.           a := 4
7.        else
8.           a := 6
9.        z := a
10. write (z)
```

## Methods (Graph Reachability on PDG)

```
1.  read (n)
2.  for I := 1 to n do
3.     a := 2
4.     if c1 then
5.        if c2 then
6.           a := 4
7.        else
8.           a := 6
9.        z := a
10. write (z)
```




## Methods (Graph Reachability on PDG)

```
1.  read (n)
2.  for I := 1 to n do
3.     a := 2
4.     if c1 then
5.        if c2 then
6.           a := 4
7.        else
8.           a := 6
9.        z := a
10. write (z)
```

**PDG**

## Methods (Graph Reachability on PDG)

- Issues with algorithm
  - Slicing criterion only for variables used in nodes
  - PDG not defined for programs with non-structured control flow—e.g., those containing exits or exceptions
  - Must construct the PDG for the entire program before slicing

## Static Backward Slicing

- Considered intraprocedural
- Studied
  - Data flow on CFG
  - Graph reachability on PDG

## Dynamic Slicing Dependence Graphs-1

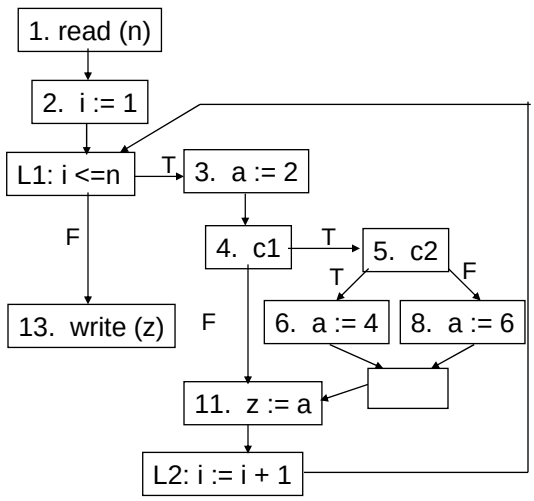
Algorithm

1. Mark *nodes* in PDG that are executed; produces reduced PDG
2. Static backward slice on reduced PDG

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**CFG**



## Dynamic Slicing Dependence Graphs-1

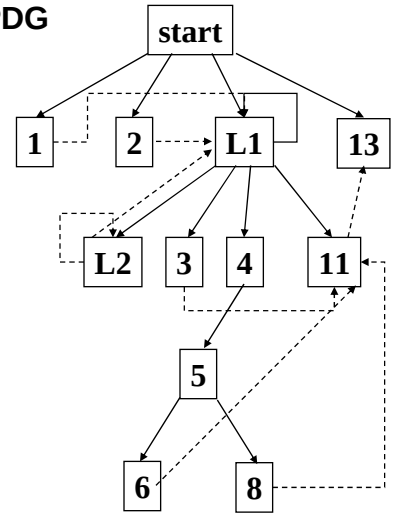
```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**PDG**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**PDG**



## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

**Example 1**

**Input** n is 1; c1, c2 both true

**Execution history** is

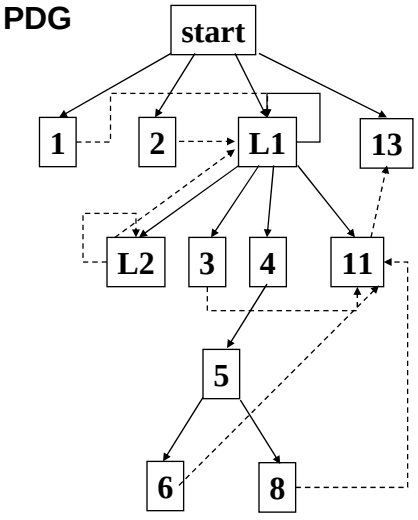
$1^1$, $2^2$, $L1^3$ ,$3^4$, $4^5$, $5^6$, $6^7$, $11^7$, $L2^8$,$L1^9$, $13^{10}$

**Criterion** <1, $13^{10}$, z>

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

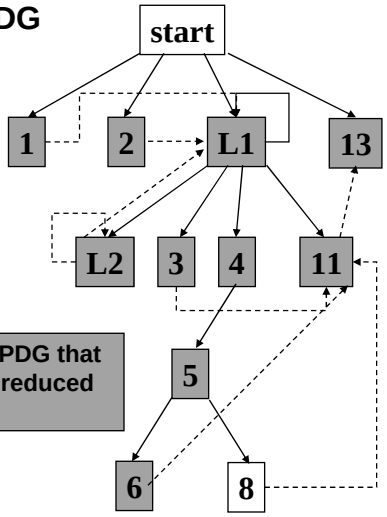
**PDG**



## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**PDG**

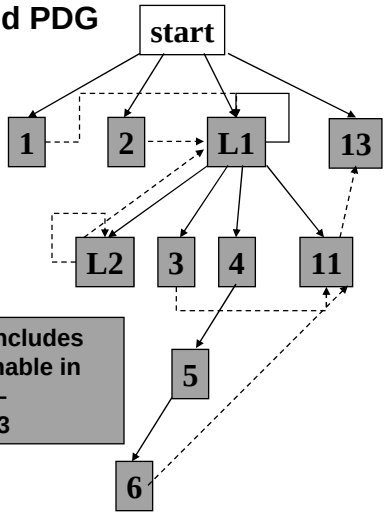



**Mark nodes in PDG that are executed→reduced PDG**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**




**Slice from 13 includes all nodes reachable in reduced PDG— 1,2,3,4,5,6,11,13**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

**Example 2**

**Input** n is 2; c1 false on first iteration, c1, c2 true on second iteration

**Execution history** is

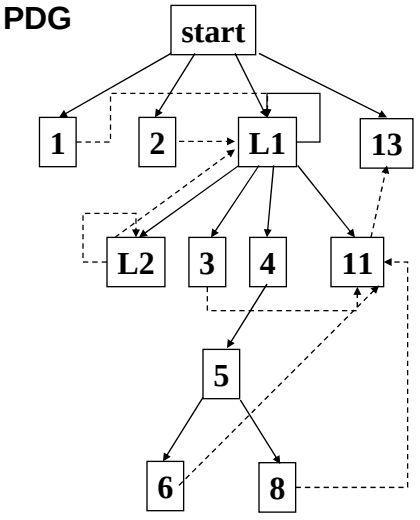
1, 2, L1, 3, 4, 11, L2, L1, 3, 4, 5, 6, 11, L2, L1,13

**Criterion** <1, 13, z>

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

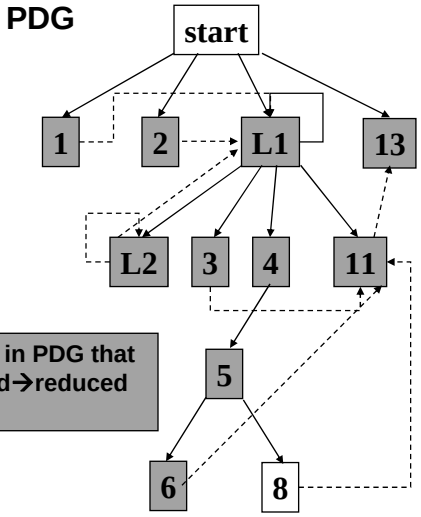
**PDG**



## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**PDG**

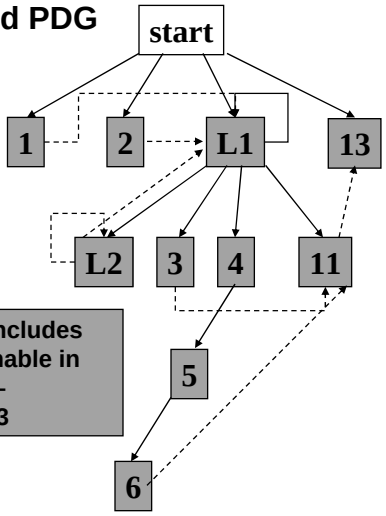



**Mark nodes in PDG that are executed→reduced PDG**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**




**Slice from 13 includes all nodes reachable in reduced PDG— 1,2,3,4,5,6,11,13**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Example 3**

**Input** n is 3; c1 false on first iteration, c1, c2 true on second iteration, c1 is true on third iteration, c2 is false on third iteration

**Execution history** is

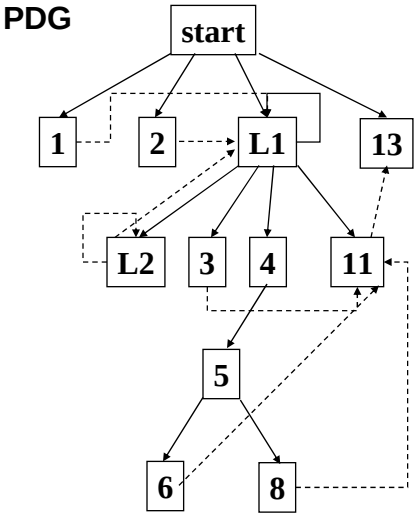
1, 2, L1, 3, 4, 11, L2, L1, 3, 4, 5, 6, 11, L2, L1, 3, 4, 5, 8, 11, L2, L1,13

**Criterion** <1, 13, z>

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

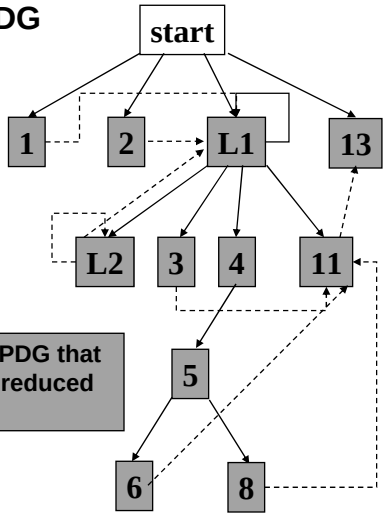
**PDG**



## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

**PDG**

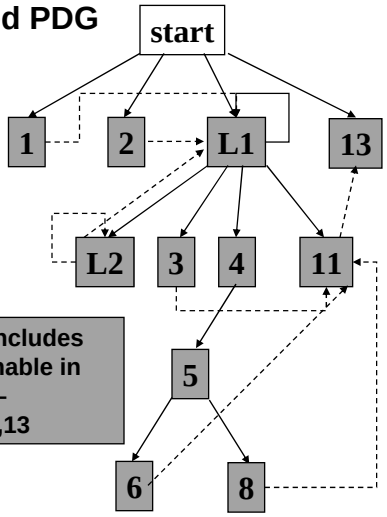



**Mark nodes in PDG that are executed→reduced PDG**

## Dynamic Slicing Dependence Graphs-1

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**




**Slice from 13 includes all nodes reachable in reduced PDG— 1,2,3,4,5,6,8,11,13**

## Dynamic Slicing Dependence Graphs-1

Algorithm

1. Mark *nodes* in PDG that are executed; produces reduced PDG
2. Static backward slice on reduced PDG

Issues ??

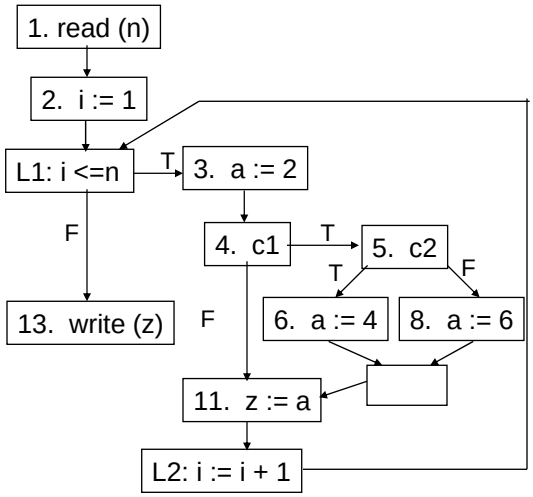
## Dynamic Slicing Dependence Graphs-2

Algorithm

1. Mark *edges* in PDG that are executed; produces reduced PDG
2. Static backward slice on reduced PDG

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```




## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Example 4**

**Input** n is 1; c1, c2 both true

**Execution history** is

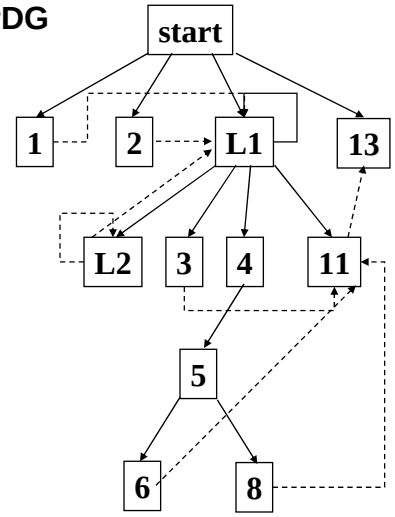
1, 2, L1 ,3, 4, 5, 6, 11, L2, L1, 13

**Criterion** <1, 13, z>

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

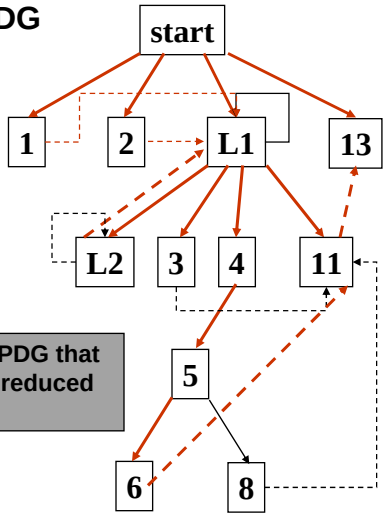
PDG




## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

PDG

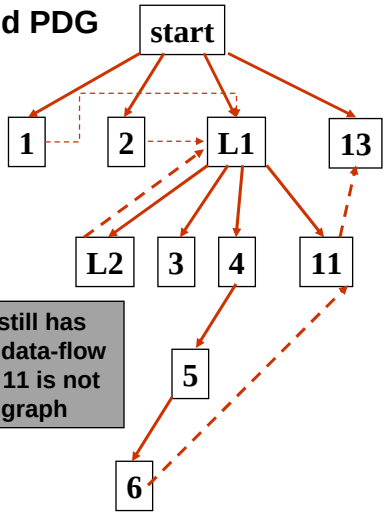



**Mark edges in PDG that are executed→reduced PDG**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**

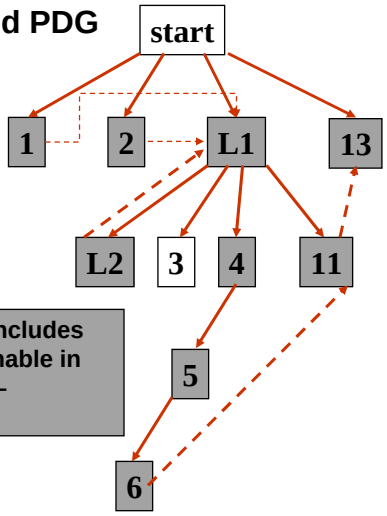



**Reduced PDG still has node 3 but the data-flow edge from 3 to 11 is not in the reduced graph**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**

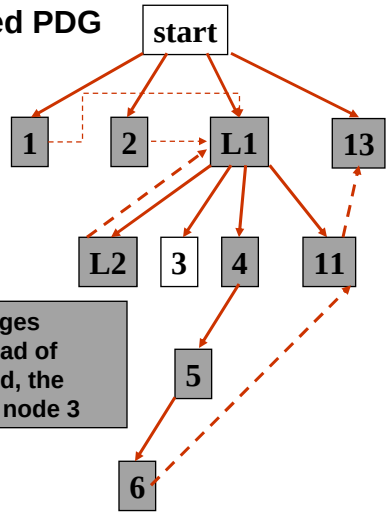



**Slice from 13 includes all nodes reachable in reduced PDG— 1,2,4,5,6,11,13**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**




**By marking edges executed instead of nodes executed, the slice excludes node 3**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Example 5**

**Input** n is 2; c1 true, c2 false on first iteration, c1, c2 true on second iteration

**Execution history** is

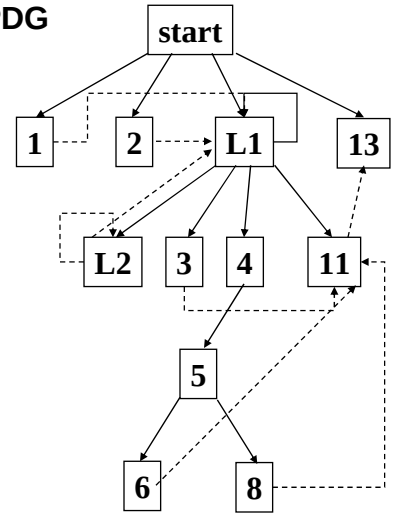
1, 2, L1, 3, 4, 5, 8,11, L2, L1, 3, 4, 5, 6, 11, L2, L1,13

**Criterion** <1, 13, z>

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

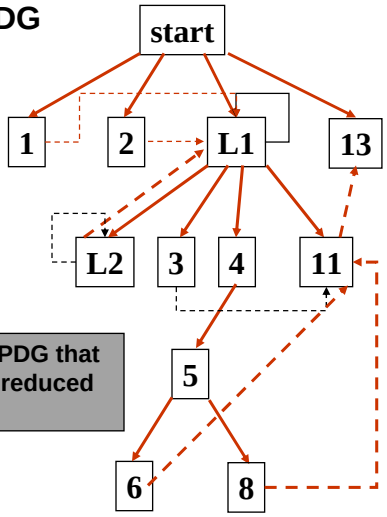
PDG




## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

PDG

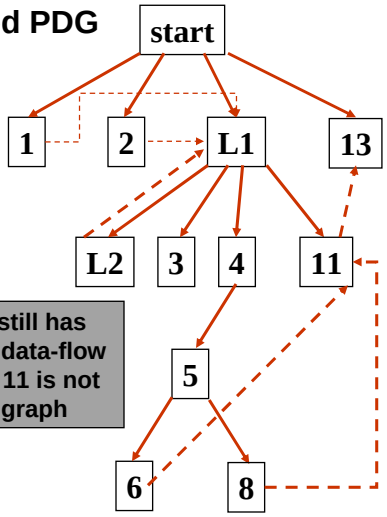



**Mark edges in PDG that are executed→reduced PDG**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**

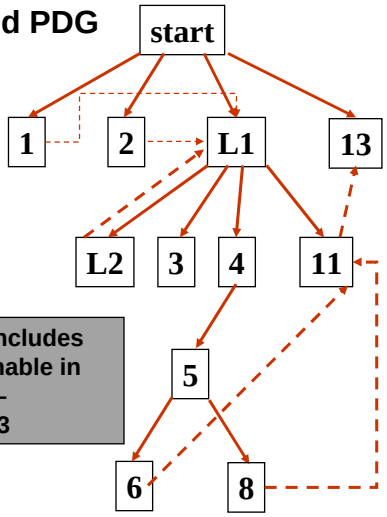



**Reduced PDG still has node 3 but the data-flow edge from 3 to 11 is not in the reduced graph**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**

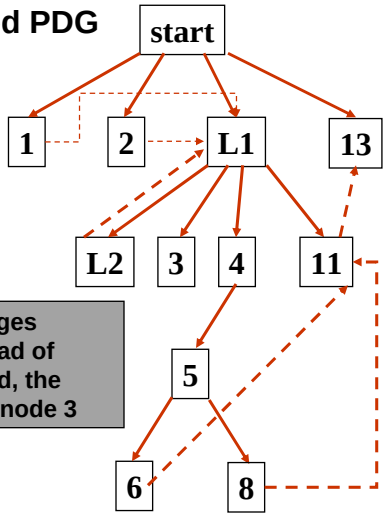



**Slice from 13 includes all nodes reachable in reduced PDG—1,2,4,5,6,8,11,13**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**

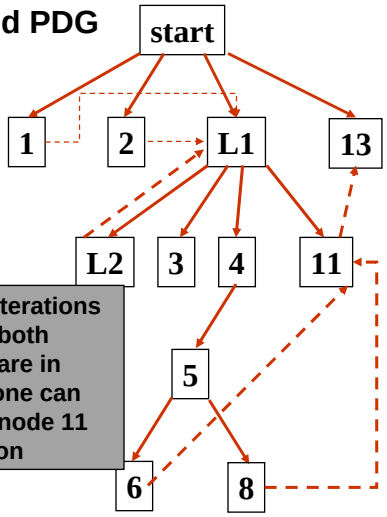



**By marking edges executed instead of nodes executed, the slice excludes node 3**

## Dynamic Slicing Dependence Graphs-2

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Reduced PDG**




**Because loop iterations are collapsed, both nodes 6 and 8 are in slice but only one can actually reach node 11 during execution**

## Dynamic Slicing Dependence Graphs-2

Algorithm

1. Mark *edges* in PDG that are executed; produces reduced PDG
2. Static backward slice on reduced PDG

Issues ??

## Dynamic Slicing Dependence Graphs-3

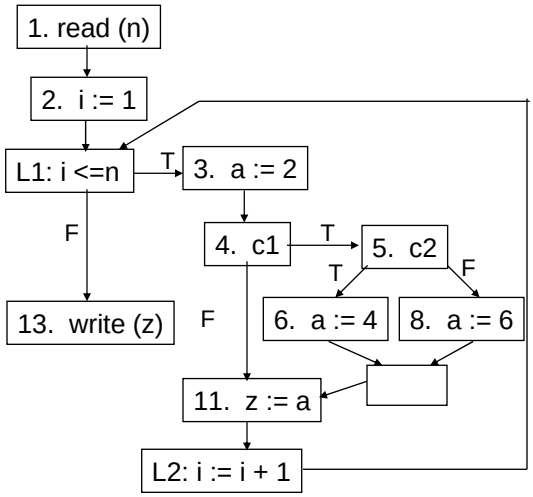
Algorithm

1. Create *dynamic dependence graph*
2. Static backward slice on reduced PDG

Issues ??

## Slicing Using Dependence Graphs-3

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```




## Slicing Using Dependence Graphs-3

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

**Example 6**

**Input** n is 2; c1 true, c2 false on first iteration, c1, c2 true on second iteration

**Execution history** is

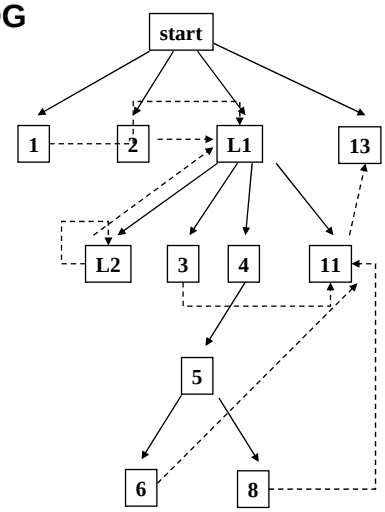
1, 2, L1, 3, 4, 5, 8,11, L2, L1, 3, 4, 5, 6, 11, L2, L1,13

**Criterion** <1, 13, z>

(Discussed in class)

## Dynamic Slicing Dependence Graphs-2

DDG




## Dynamic Slicing Dependence Graphs-3

Algorithm

1. Create *dynamic dependence graph*
2. Static backward slice on reduced PDG