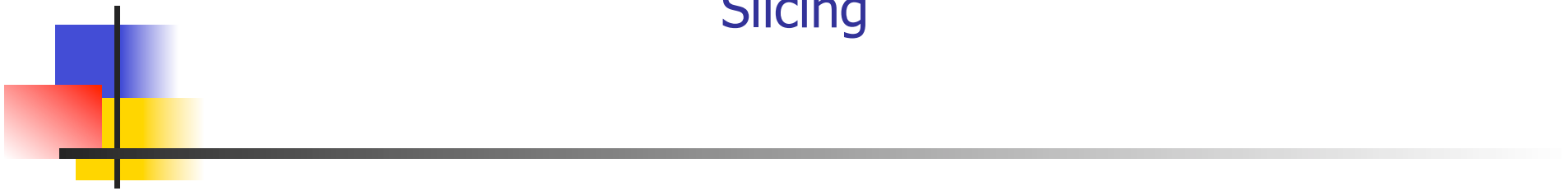
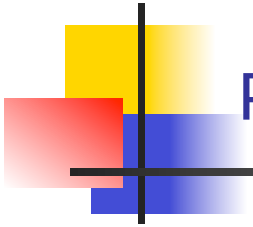


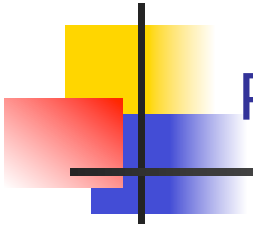
# Slicing

Chapter 9



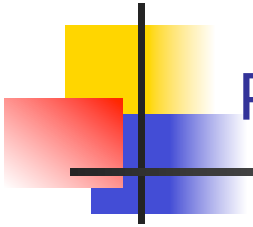
# Program slice

- **What is a program slice?**



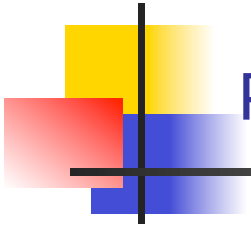
# Program slice – Informally

- **What is a program slice?**
  - **A program slice is a set of program statements that contributes to or affects the value of a variable at some point in a program**



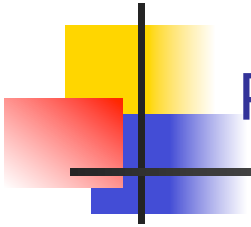
# Program slice – Formally

- **What is a program slice?**
- Given a program **P** and a set of variables **V** in **P** and a statement or statement fragment **n**
  - **A slice S(V, n) is**
    - **A set of node numbers in a program graph**
    - **The set of all statements and statement fragments in P prior to the node n that contribute to the values of variables in V at node n.**
  - **Prior to is a dynamic execution time notion**



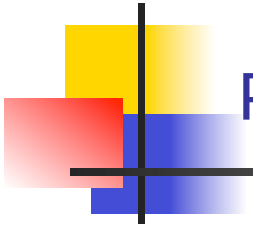
# Program slice – Point of

- Analyze a program by focusing on parts of interest, disregarding uninteresting parts
  - **The point of slices is to separate a program into components that have a useful functional meaning**
  - **Ignore those parts that do not contribute to the functional meaning of interest**
  - **Cannot do this with du-paths, as slices are not simply sequences of statements or statement fragments**



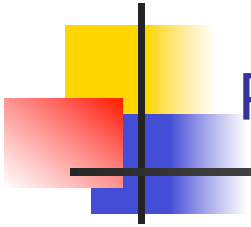
# Program slice – meaning of "contributes to"

- Refine the meaning of usage and defining nodes
  - **P-use – used in a decision predicate**
  - **C-use – used in a computation**
  - **O-use – used for output**
  - **L-use – used for location (pointers, subscripts)**
  - **I-use – used for iteration (loop counters, loop indices)**
  - **I-def – defined by input**
  - **A-def – defined by assignment**
    - **Textbook excludes all non-executable statements such as variable declarations**



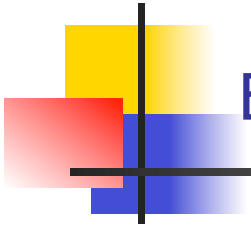
# Program slide – meaning of "contributes to" – 2

- What to include in S(V,n)?
  - **Consider a single variable v**
    - **Include all I-def, A-def**
    - **Include any C-use, P-use of v, if excluding it would change the value of v**
    - **Include any P-use or C-use of another variable, if excluding it would change the value of v**



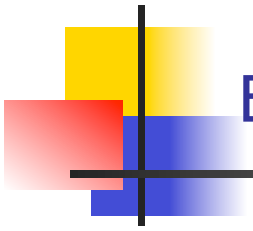
# Program slide – meaning of "contributes to" – 3

- What to include in S(V,n)?
  - **Consider a single variable v**
    - **L-use and I-use**
      - Inclusion is a judgment call, as such use does cause problems
    - **Exclude all non-executable nodes such as variable declarations**
      - If a slice is not to be compilable
    - **Exclude O-use, as does not change the value of v**



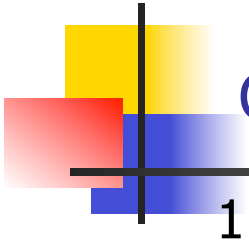
# Example 1 – What is S(sum,8)?

```
1 int i;
2 int sum = 0;
3 int product = 1;
4 for(i = 0; i < N; ++i) {
5   sum = sum + i;
6   product = product * i;
7 }
8 write(sum);
9 write(product);
```



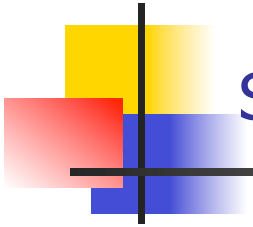
# Example 1 – S(sum,8)

```
1 int i;
2 int sum = 0;
4 for(i = 0; i < N; ++i) {
5   sum = sum + i;
7 }
8 write(sum);
```



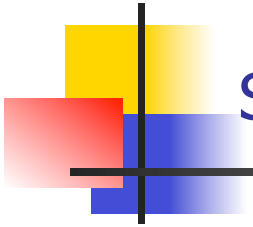
# Class Exercise

```
1   program Example()
2   var staffDiscount, totalPrice, finalPrice, discount, price
3   staffDiscount = 0.1
4   totalPrice = 0
5   input(price)
6   while(price != -1) do
7      totalPrice = totalPrice + price
8      input(price)
9   od
10  print("Total price: " + totalPrice)
11  if(totalPrice > 15.00) then
12     discount = (staffDiscount * totalPrice) + 0.50
13  else
14     discount = staffDiscount * totalPrice
15  fi
16  print("Discount: " + discount)
17  finalPrice = totalPrice - discount
18  print("Final price: " + finalPrice)
19  endprogram
```



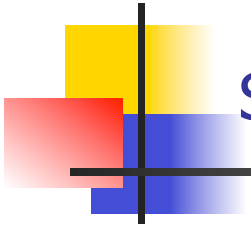
# Slice style & technique

- Make slices on one variable
  - **Slices with more variables are super sets of a one variable case**
  - **Do not make a slice S(V, n) where the variables of interest are not in node n**



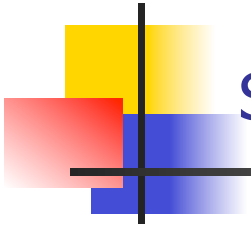
# Slice style & technique – 2

- Make slices for all A-def nodes
- Make slices for all P-use nodes
  - **Very useful in decision intensive programs**
- Try to make slices compilable
  - **Means including declarations and compiler directives**
  - **Such slices become executable and more easily tested**



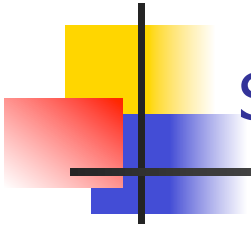
# Slice style & technique – 3

- Avoid slices on C-use
  - **They tend to be redundant**
- Avoid slices on O-use
  - **They are the union of all the A-def and I-def slices**
    - **Dramatically increases test effort**



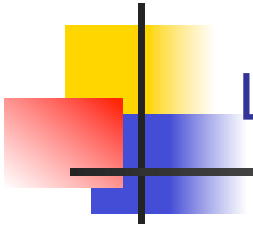
# Slice style & technique – 4

- Relative complement of slices can have diagnostic value
  - **If you have difficulty at a part, divide the program into two parts**
  - **If the error does not lie in one part, then it must be in the relative complement**



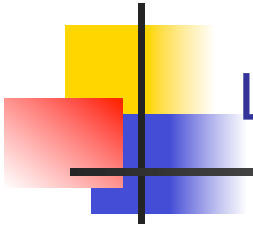
# Slice style & technique – 5

- Slices and DD-paths have a many-to-many relationship
  - **Nodes in one slice may be in many DD-paths, and nodes in one DD-path may be in many slices**
  - **Sometimes well-chosen relative complement slices can be identical to DD-paths**
- Developing a lattice of slices can improve insight in potential trouble spots



# Lattice

- **What is a lattice?**



# Lattice – 2

- **What is a lattice?**
  - **A directed acyclic graph**
  - **Shows "contained-in" relationships**
    - See Figures 9.9 & 9.10 and Class Exercise