# *Program Slicing*

A Survey of Program Slicing,

Binkley and Gallagher

*Advances in Computers, 1996*

# Program Slicing

What statements effect the value of ____ at line ____? (line numbers for discussion only.)

```
1.  c = 4;
2.  b = c;
3.  a = b + c;
4.  d = a + c;
5.  f = d + b;
6.  a = d + 8;
7.  b = f + 30;
8.  a = b + c:
```

# How about this one?

```
1.  a = ….;
2.  b = …;
3. if (a <= b)
4.           x = a + b;
5. else
6.           y = a – b ;
7.  ….
```

# *And this one?*

```
while(…) {
    e = d;
    d = c;
    c = b ;
    b = a;
 }
```

# *And?*

```
c = 0;
while (true) {
    c = 1;
}
c = 2;
```

# *Minimal???*

```
input x;
if (x) {
        // nothing involving x here..

         x = 1;
}
else x = 2;
```

# *What about “a”?*

```
•   a = 5;
•   while (p(k)) {
•    if (q(c)) then {
•                 b = a;
•                 x = 1;
•          } else { c = b;
•                    y = 2; }
•          k = k + 1;
•   }
•   z = x + y;
```

# “The Classic”

```
»   #define YES 1
»   #define NO 0
»   #include <stdio.h>
»   main() {
»           int c, nl, nw, nc, inword;
»           inword = NO;
»           nl = 0;
»           nw = 0;
»           nc = 0;
»           c = getchar();
»           while ( c != EOF ) {
»                nc = nc + 1;
»                if ( c == '\n') nl = nl + 1;
»                if ( c == ' ' || c == '\n' || c == '\t')
»                         inword = NO;
»                else if ( inword == NO )
»                      { inword = YES;
»                        nw = nw + 1; }
»                 c = getchar();
»            }
»            printf("%d \n", nl);
»            printf("%d \n", nw);
»            printf("%d \n", nc);
»   }
```

# Sub-Classic 1

```
– #include <stdio.h>
– main() {
–           int c, nl, nw, nc, inword; /* !! ?? */
–           nc = 0;
–           c = getchar();
–           while ( c != EOF ) {
–                 nc = nc + 1;
–                 c = getchar();
–             }
–          printf("%d \n", nc);
–     }
```

# Sub-Classic 2

```
– #include <stdio.h>
–  main() {
–        int c, nl, nw, nc, inword;
–        nl = 0;
–        c = getchar();
–        while ( c != EOF ) {
–            if ( c == '\n') nl = nl + 1;
–           c = getchar();
–         }
–       printf("%d \n", nl);
–   }
```

# *Sub-Classic 3*

```
• #include <stdio.h>
• #define YES 1
• #define NO 0
• main() {
•         int c, nl, nw, nc, inword;
•         inword = NO;
•         nw = 0;
•         c = getchar();
•         while ( c != EOF ) {
•          if ( c == ' ' || c == '\n' || c == '\t')
•                       inword = NO;
•              else if ( inword == NO )
•                    { inword = YES;
•                     nw = nw + 1; }
•               c = getchar();
•          }
•         printf("%d \n", nw);
•     }
```

# Sub-Classic 4

```
– #include <stdio.h>
– #define YES 1
– #define NO 0
– main() {
–         int c, nl, nw, nc, inword;
–         inword = NO;
–         c = getchar();
–         while ( c != EOF ) {
–                 if ( c == ' ' || c == '\n' || c == '\t')
–                       inword = NO;
–              else if ( inword == NO )
–                    { inword = YES;
–                     }
–               c = getchar();
–          }
–       }
```

# Sub-Classic 5

```
#include <stdio.h>
#define YES 1
#define NO 0
main() {
    int c, nl, nw, nc, inword;
    c = getchar();
    while ( c != EOF ) {
        c = getchar();
     }
 }
```

# *Files and Functions*

```
int rufus,toby;
main ( )  {
  int polar,watergate;
  polar = 1;
  rufus = 2;
  toby = 3;
  watergate = 4;
  ride (toby );
}
print (int critter) {
   printf ("%d",critter );
}
```

```
extern int rufus, toby ;
ride  (int horse) {
  int mule,donkey;

  mule = horse;
  print (rufus );
  donkey = toby + horse;
  print (horse );
  horse++;
  rufus = donkey;
  toby = mule;
}
```

# *reference variables…*

```
»    int cond();  //  something that returns T   F
»    main  (  ) {
»        int w, x, y, z;
»        int *e, *f, *g, *h, *i, *j ;
»        int **b, **c, **d;
»        int ***a;
»         a = cond ( ) ?  (cond ( ) ? &b : &c ) : &d;
»         b = cond ( ) ? &e : &f;
»         c = cond ( ) ? &g : &h;
»         d = cond ( ) ? &i : &j;
»         e = cond ( ) ? &i : &j;
»         f = &x;
»         g = &y;
»         h = &z;
»         i = &w;
»         j = cond ( ) ? &w : &z;
»         /* now assign to w, x, y or z */
»       }
```

# *Preliminaries*

- **State:** Variables => Values
- **Statement:** State => State
- **Program:** Sequence of statements
- **Slicing criterion <v,n> (of program P)**
  - **v** variable
  - **n** statement number

# *Finally!*

- **Program Slice, S,  (of P) at <v,n>**
  - **Only those statements needed to *capture the behaviour* of v at n.**
- **Every program has itself as a slice on any criteria.**
- So what does “capture the behaviour” mean?

# *Formally*

- **Executable Program Slice, S, (of P)**
  - S can obtained from deleting 0 or more statements from P
  - If P halts on input I, then the value of **v** at statement **s** each time **s** is executed is the same in P and S.
- This is **static**
- This is also **backward**
- Look at slices from last lecture

# But..

- Slices do not need to be executable
  - Executable / non-executable
  - Compliable or not.
- Slices do not need to be static
  - Static / Dynamic
  - All inputs or 1 input
- Slices do not need to be backward
  - Forward / backward
  - Or both!

# *Background (jargon)*

- Graph: N(odes); E(dges). Digraphs
- “Immediate predecessor”
- “Immediate successor”
- Path
- Dominators
- Restrict to single-entry, single-exit

# *More…*

- Change programs into graphs
  - Statements are nodes
  - Edges show control

- At each node:
  - **Refs(n)**, the variables referenced at n
  - **Defs(s),** the variables defined at n

# *How to compute relevant sets (and slices)*

- .
- ..
- …
- ….
- …..
- …….
- more???

# Compute Relevant Sets for <a, 8>

| n | stmt | defs(n) | refs(n) | relevant |
|---|---|---|---|---|
| 1. | c = 4; | c | | |
| 2. | b = c; | b | c | |
| 3. | a = b + c; | a | b, c | |
| 4. | d = a + c; | d | a, c | |
| 5. | f = d + b; | f | d, b | |
| 6. | a = d + 8; | a | ____ | |
| 7. | b = f + 30; | b | ____ | |
| 8. | a = b + c: | __ | ____ | |

# *Computing (and using) Control Sets*

- control(n): the set of predicate statements that directly control the execution of statement n.
- Whenever **n** is added to a slice, so are the members of control(n).
- **and** <refs(control(n), control(n)> is added to criteria.
- At joins, relevant(k) = relevant(succ(k))

# *Control Sets*

| **n** | **stmt** | **defs(n)** | **refs(n)** | **control(n)** | **rel(n)** |
|---|---|---|---|---|---|
| 1. | b = 4; | | | | |
| – | c = 2; | | | | |
| – | d = 3; | | | | |
| – | a = d; | | | | |
| 5. | if (a) then | | | | |
| 6. | d = b + d | | | | |
| 7. | c = b + d | | | | |
| – | else | | | | |
| – | b = b + 1 | | | | |
| – | d = d + 1 | | | | |
| – | endif | | | | |
| – | a = b + c; | | | | |
| – | print a | | | | |

| | def | ref | cont | rel | slice |
|---|---|---|---|---|---|
| while(X) { | | | | | |
| e = d; | | | | | |
| d = c; | | | | | |
| c = b ; | | | | | |
| b = a; | | | | | |
| } | | | | | |

# *Loops*

- For loops, iterate until slice & relevant sets stabilize.
- What is (worst case) running time of this approach?
  - Call it homework…..
- What about **return, break, exit, goto**?
  - Call it more homework…..
- And procedure/method calls?