Program Analysis via Graph Reachability

Taken from Thomas Reps'
PLDI 2000 Tutorial

Software Eng. Applications

- Program understanding
- Reengineering
- Testing & Analysis

Program Slicing

- The backward slice w.r.t variable *v* at program point *p*
  - The program subset that may influence the value of variable *v* at point *p*.
- The forward slice w.r.t variable *v* at program point *p*
  - The program subset that may be influenced by the value of variable *v* at point *p*.

Weiser's Slicing Algorithm

- How does Weiser's definition of a slice differ from Reps'?

Backward Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
         sum = sum + i;
         i = i + 1;
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

Backward slice with respect to "printf("%d\n",i)"

Backward Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
         sum = sum + i;
         i = i + 1;
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

Backward slice with respect to "printf("%d\n",i)"

Slice Extraction

```
int main() {

     int i = 1;
     while (i < 11) {

         i = i + 1;
     }

     printf("%d\n",i);
}
```

Backward slice with respect to "printf("%d\n",i)"

Forward Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
         sum = sum + i;
         i = i + 1;
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

Forward slice with respect to "sum = 0"

Forward Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
         sum = sum + i;
         i = i + 1;
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

Forward slice with respect to "sum = 0"

What Are Slices Useful For?

- Understanding Programs
  - What is affected by what?
- Restructuring Programs
  - Isolation of separate "computational threads"
- Program Specialization and Reuse
  - Slices = specialized programs
  - Only reuse needed slices
- Program Differencing
  - Compare slices to identify changes
- Testing
  - What new test cases would improve coverage?
  - What regression tests must be rerun after a change?

Line-Character-Count Program

```
void line_char_count(FILE *f) {
     int lines = 0;
     int chars;
     BOOL eof_flag = FALSE;
     int n;
     extern void scan_line(FILE *f, BOOL *bptr, int *iptr);
     scan_line(f, &eof_flag, &n);
     chars = n;
     while(eof_flag == FALSE){
         lines = lines + 1;
         scan_line(f, &eof_flag, &n);
         chars = chars + n;
     }
     printf("lines = %d\n", lines);
     printf("chars = %d\n", chars);
}
```

Character-Count Program

```
void char_count(FILE *f) {
     int lines = 0;
     int chars;
     BOOL eof_flag = FALSE;
     int n;
     extern void scan_line(FILE *f, BOOL *bptr, int *iptr);
     scan_line(f, &eof_flag, &n);
     chars = n;
     while(eof_flag == FALSE){
         lines = lines + 1;
         scan_line(f, &eof_flag, &n);
         chars = chars + n;
     }
     printf("lines = %d\n", lines);
     printf("chars = %d\n", chars);
}
```

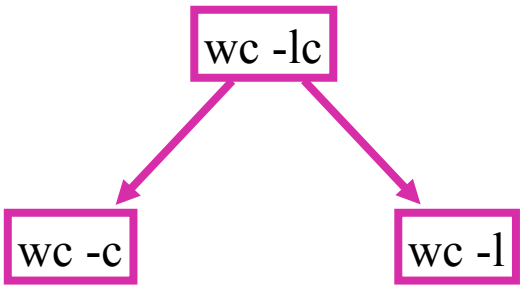
Line-Character-Count Program

```
void line_char_count(FILE *f) {
     int lines = 0;
     int chars;
     BOOL eof_flag = FALSE;
     int n;
     extern void scan_line(FILE *f, BOOL *bptr, int *iptr);
     scan_line(f, &eof_flag, &n);
     chars = n;
     while(eof_flag == FALSE){
         lines = lines + 1;
         scan_line(f, &eof_flag, &n);
         chars = chars + n;
     }
     printf("lines = %d\n", lines);
     printf("chars = %d\n", chars);
}
```

Line-Count Program

```
void line_count(FILE *f) {
     int lines = 0;
     int chars;
     BOOL eof_flag = FALSE;
     int n;
     extern void scan_line2(FILE *f, BOOL *bptr, int *iptr);
     scan_line2(f, &eof_flag    );
     chars = n;
     while(eof_flag == FALSE){
         lines = lines + 1;
         scan_line2(f, &eof_flag    );
         chars = chars + n;
     }
     printf("lines = %d\n", lines);
     printf("chars = %d\n", chars);
}
```

Specialization Via Slicing



How are Slices Computed?

- Reachability in a Dependence Graph
  - Program Dependence Graph (PDG)
    - Dependences within one procedure
    - *Intra*procedural slicing is reachability in one PDG
  - System Dependence Graph (SDG)
    - Dependences within entire system
    - *Inter*procedural slicing is reachability in the SDG

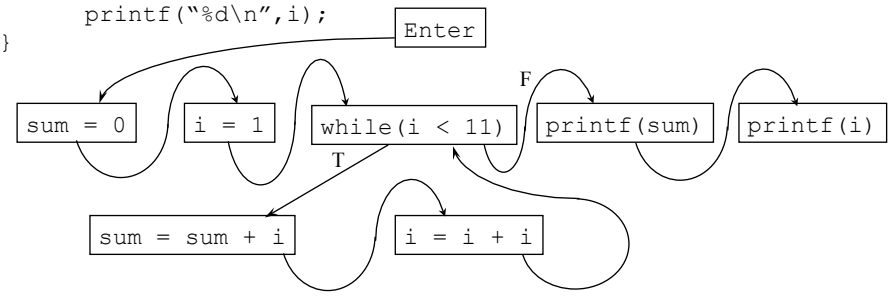
How is a PDG Created?

- Control Flow Graph (CFG)
- PDG is union of:
  - Control Dependence Graph
  - Flow Dependence Graph

  computed from CFG

Control Flow Graph

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```



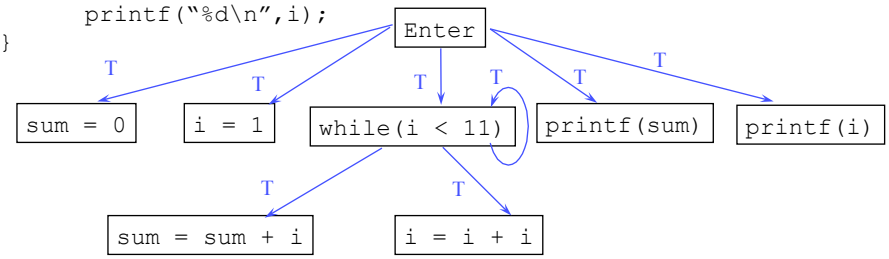
Control Dependence Graph

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```

Control dependence

$p \xrightarrow{T} q$ q is reached from p if condition p is true (T), not otherwise.

$p \xrightarrow{F} q$ Similar for false (F).

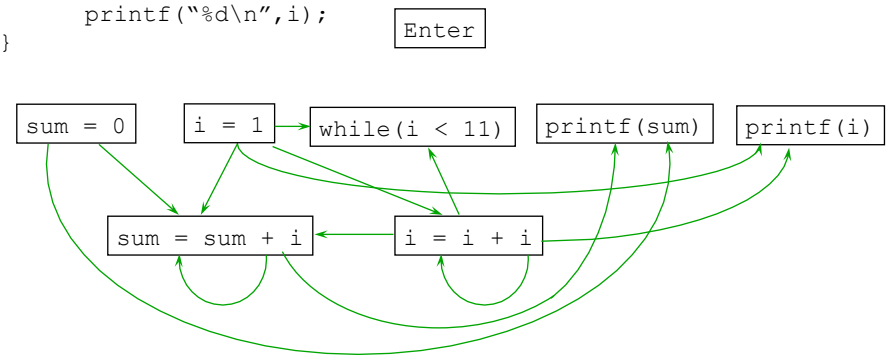


Flow Dependence Graph

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```

Flow dependence

$p \rightarrow q$ Value of variable assigned at p may be used at q.

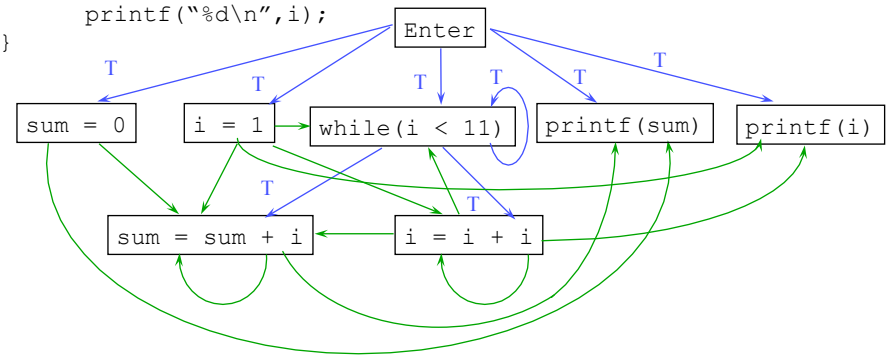


Program Dependence Graph (PDG)

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```

Control dependence

Flow dependence

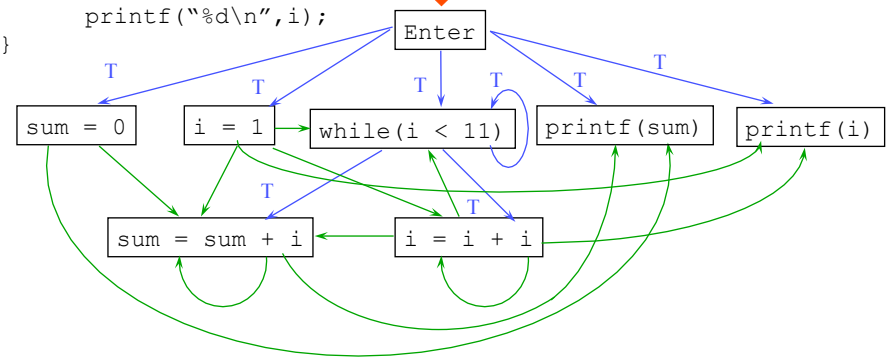


Program Dependence Graph (PDG)

```
int main() {
      int i = 1;
      int sum = 0;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```

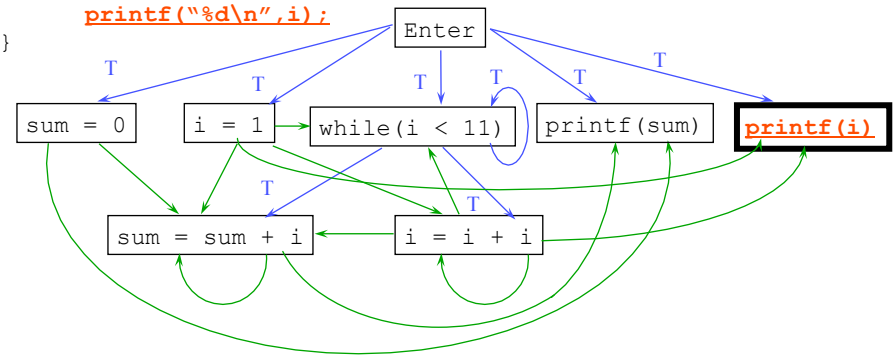
Opposite Order

Same PDG



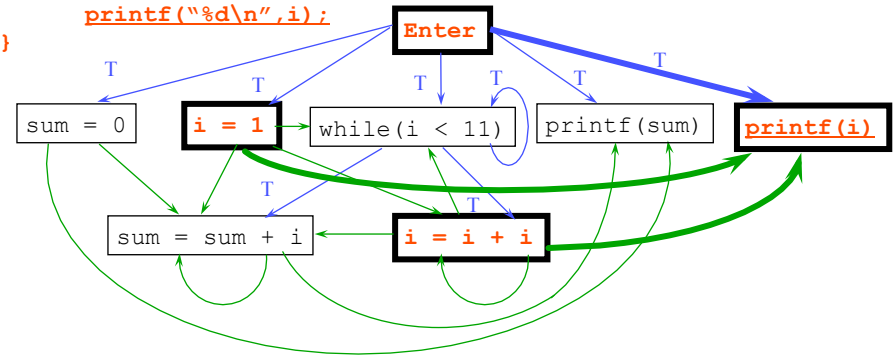
Backward Slice

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```



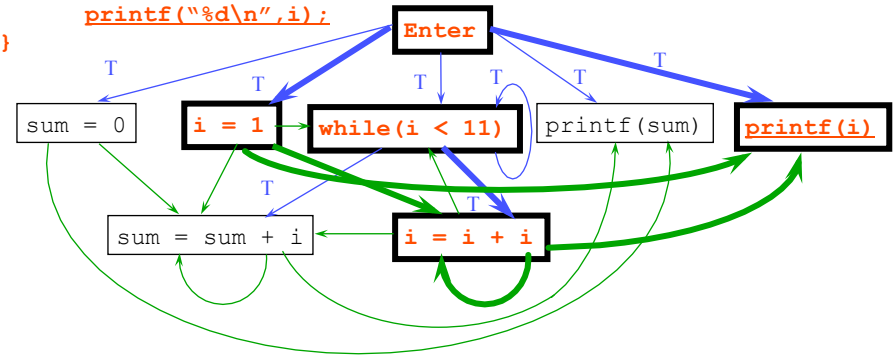
Backward Slice (2)

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```



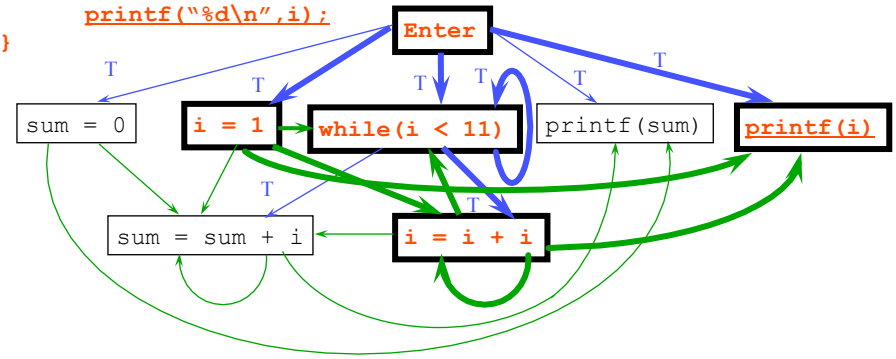
Backward Slice (3)

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```



Backward Slice (4)

```
int main() {
      int sum = 0;
      int i = 1;
      while (i < 11) {
            sum = sum + i;
            i = i + 1;
      }
      printf("%d\n",sum);
      printf("%d\n",i);
}
```



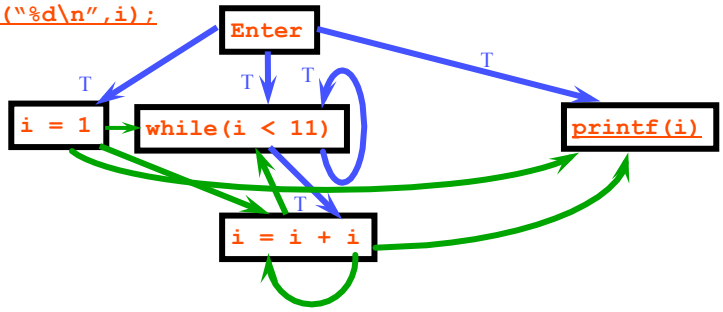
Slice Extraction

```
int main() {

      int i = 1;
      while (i < 11) {

           i = i + 1;
      }

      printf("%d\n",i);
}
```



*Inter*procedural Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
          sum = add(sum,i);
          i = add(i,1);
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

```
int add(int x, int y) {
     return x + y;
}
```

Backward slice with respect to "printf("%d\n",i)"

*Inter*procedural Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
          sum = add(sum,i);
          i = add(i,1);
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

```
int add(int x, int y) {
     return x + y;
}
```

Backward slice with respect to "printf("%d\n",i)"

*Inter*procedural Slice

```
int main() {
     int sum = 0;
     int i = 1;
     while (i < 11) {
          sum = add(sum,i);
          i = add(i,1);
     }
     printf("%d\n",sum);
     printf("%d\n",i);
}
```

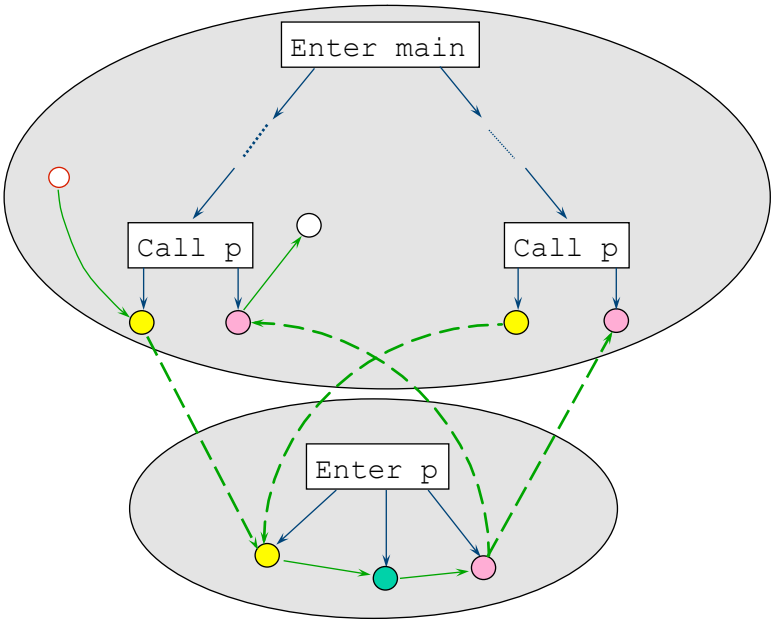
```
int add(int x, int y) {
     return x + y;
}
```

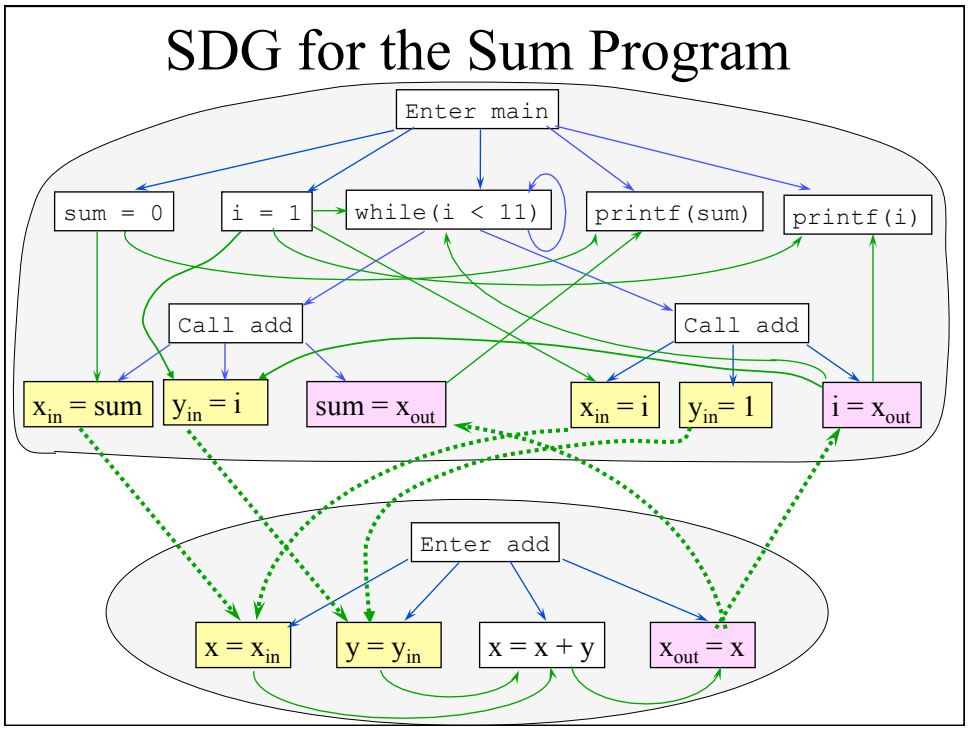
Superfluous components included by Weiser's slicing algorithm [TSE 84]
Left out by algorithm of Horwitz, Reps, & Binkley [PLDI 88; TOPLAS 90]

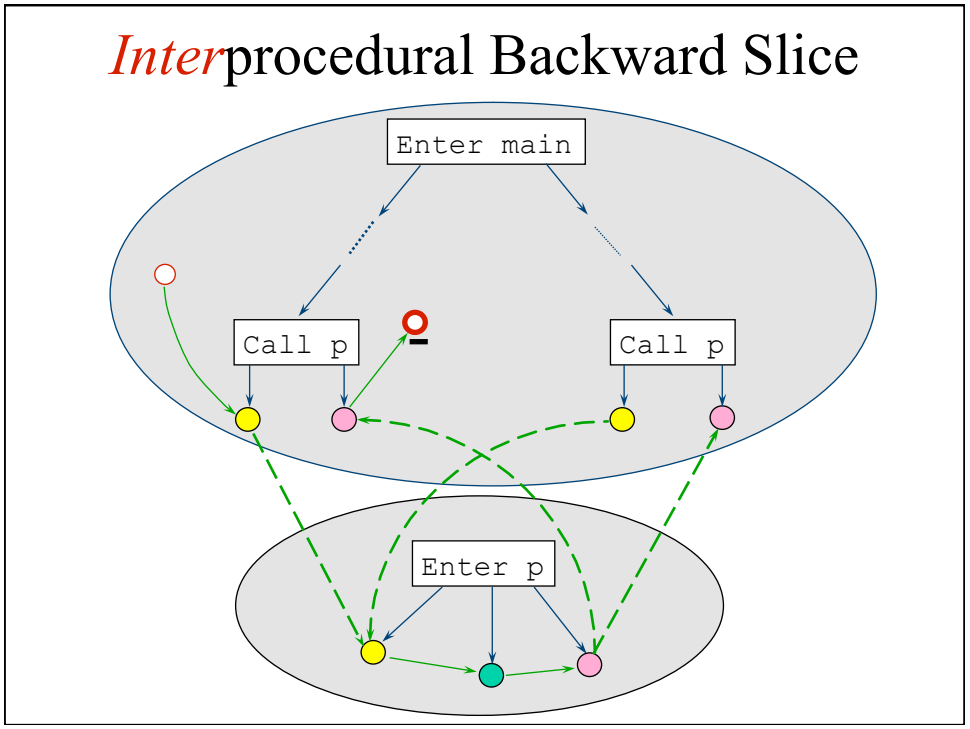
How is an SDG Created?

- Each PDG has nodes for
  - entry point
  - procedure parameters and function result
- Each call site has nodes for
  - call
  - arguments and function result
- Appropriate edges
  - entry node to parameters
  - call node to arguments
  - call node to entry node
  - arguments to parameters

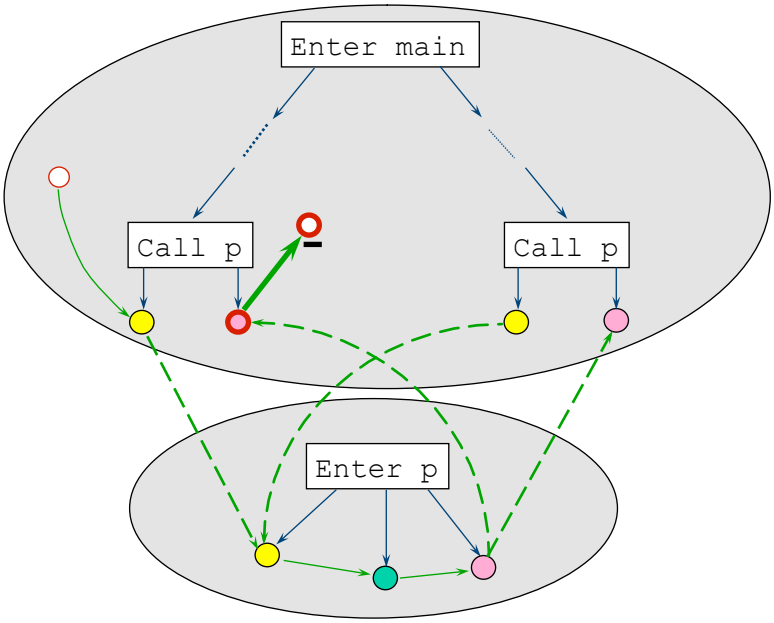
System Dependence Graph (SDG)



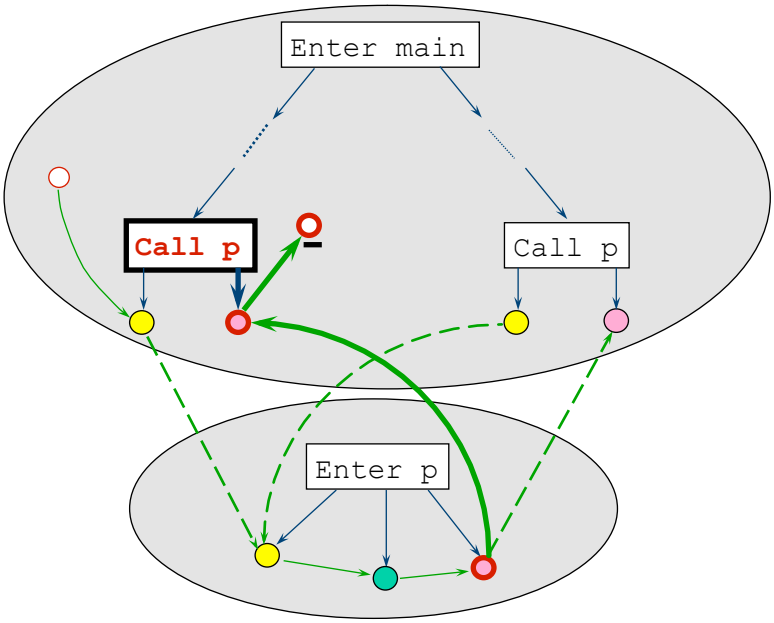

SDG for the Sum Program
Enter main
sum = 0
i = 1
while(i < 11)
printf(sum)
printf(i)
Call add
Call add
x_in = sum
y_in = i
sum = x_out
x_in = i
y_in = 1
i = x_out
Enter add
x = x_in
y = y_in
x = x + y
x_out = x


Interprocedural Backward Slice
Enter main
Call p
Call p
Enter p

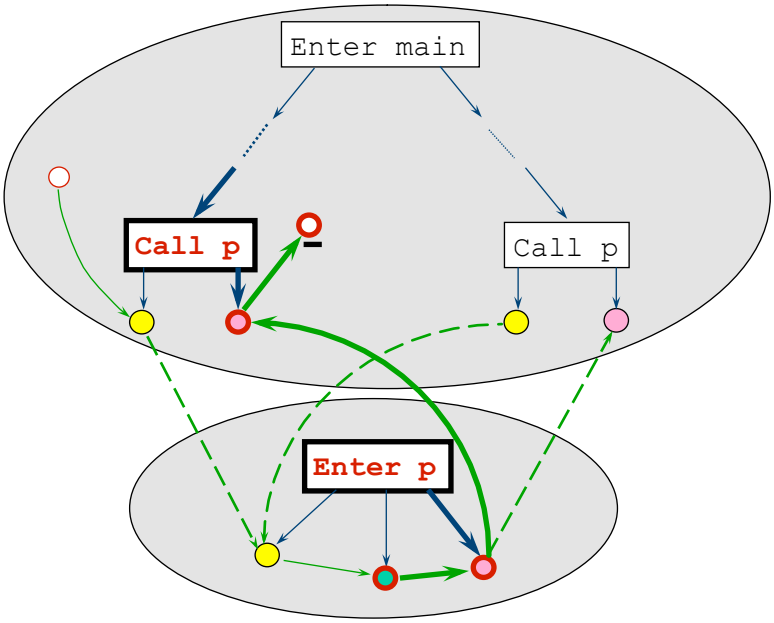
*Inter*procedural Backward Slice (2)



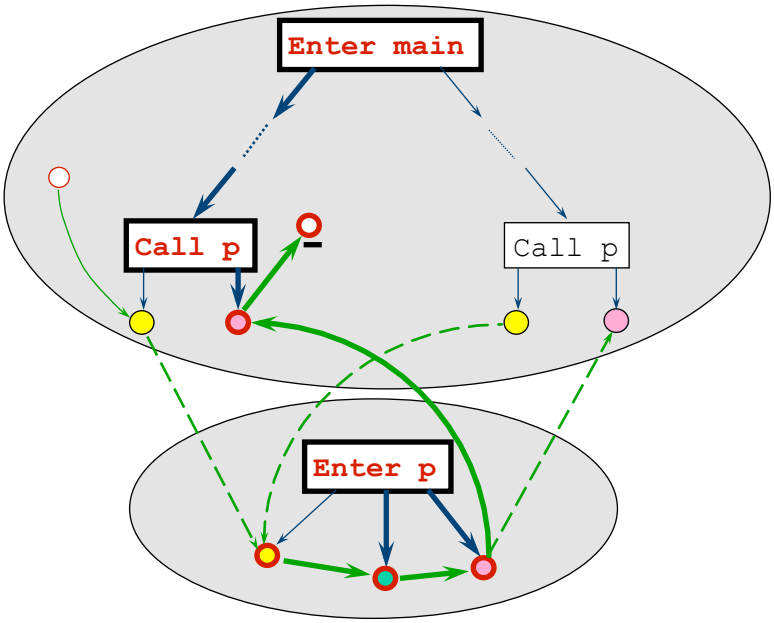
*Inter*procedural Backward Slice (3)



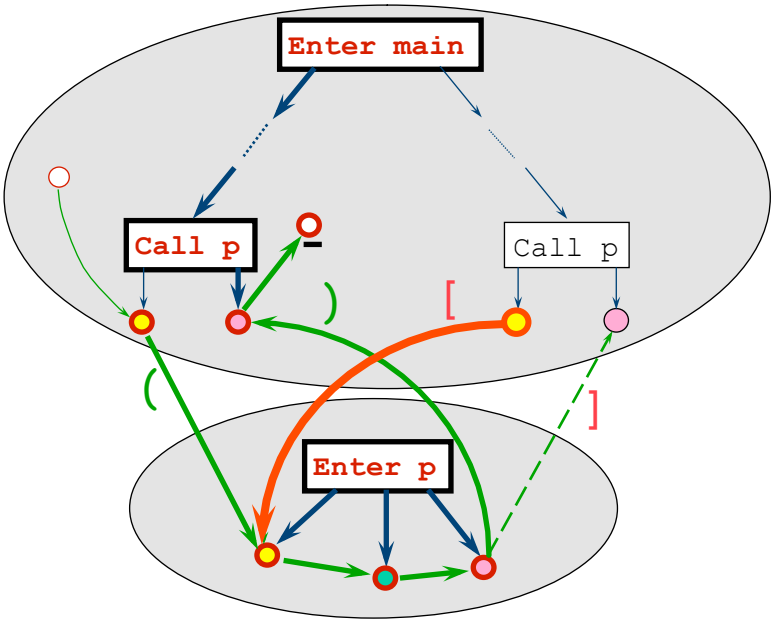
*Inter*procedural Backward Slice (4)



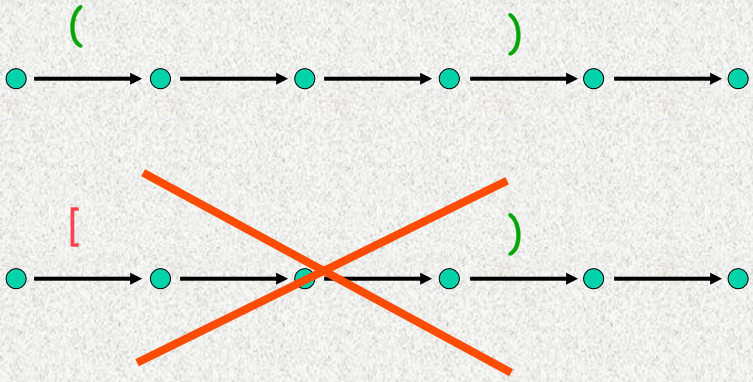
*Inter*procedural Backward Slice (5)



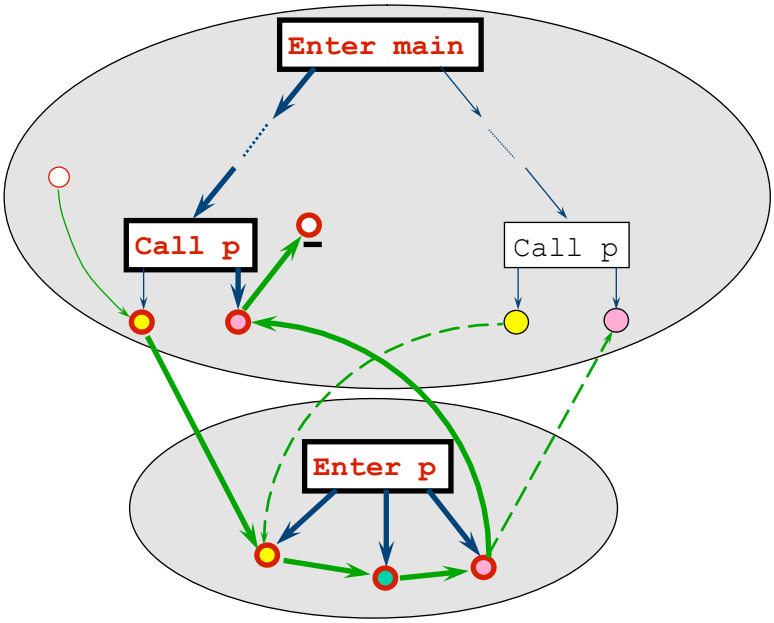
*Inter*procedural Backward Slice (6)



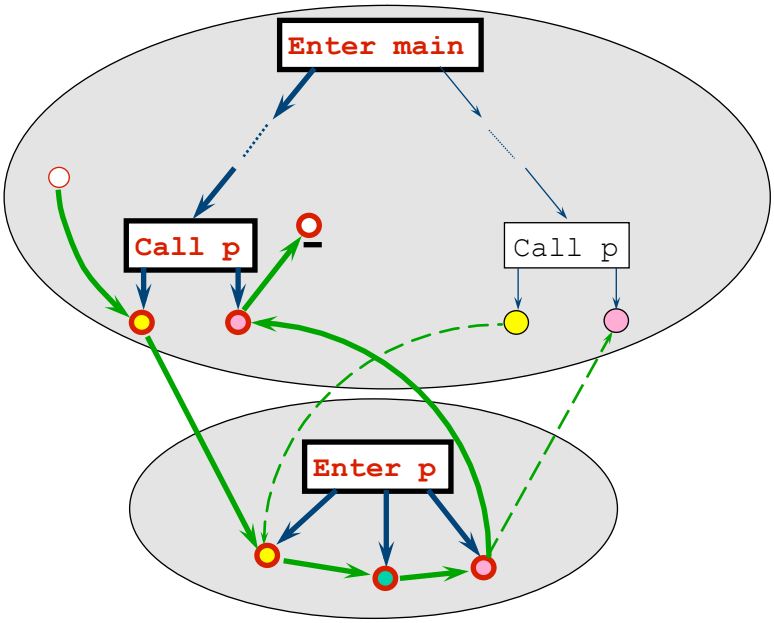
Matched-Parenthesis Path

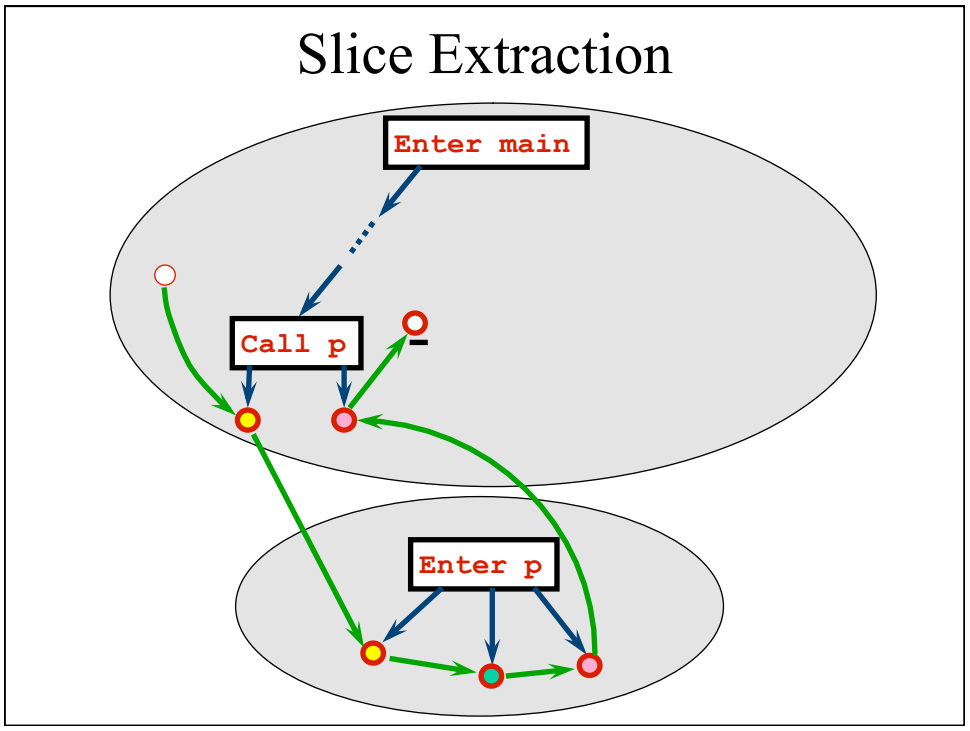


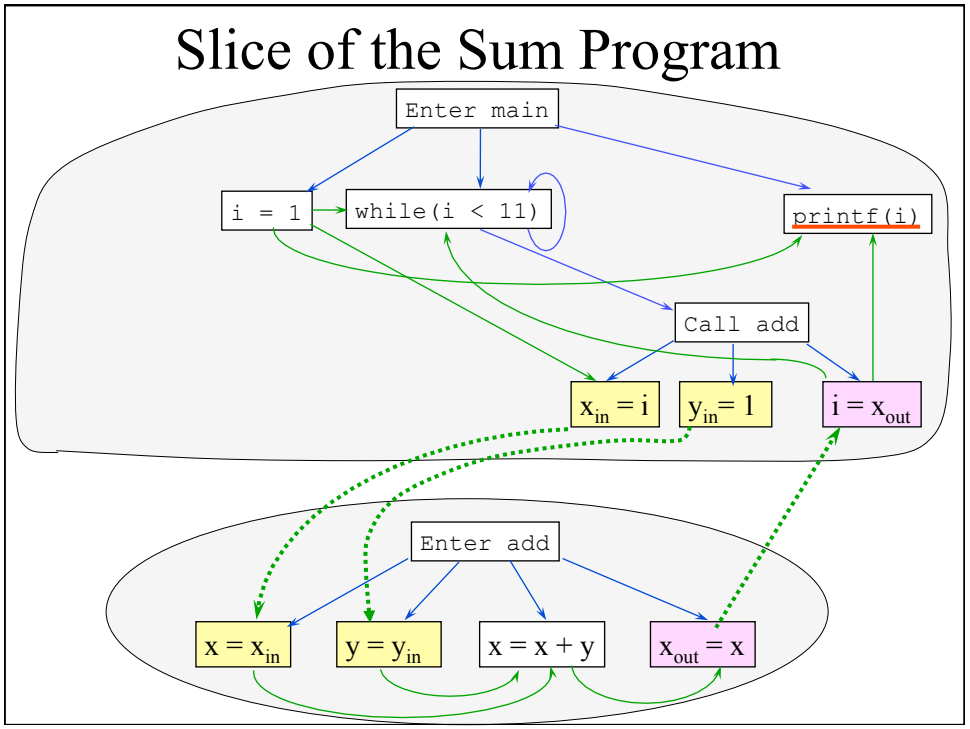
*Inter*procedural Backward Slice (6)



*Inter*procedural Backward Slice (7)




Slice Extraction
Enter main
Call p
Enter p


Slice of the Sum Program
Enter main
i = 1
while(i < 11)
printf(i)
Call add
$x_{in} = i$
$y_{in} = 1$
$i = x_{out}$
Enter add
$x = x_{in}$
$y = y_{in}$
x = x + y
$x_{out} = x$

References

- Reps, Program analysis via graph reachability, Inf. and Softw. Tech. 98

References

- Slicing, chopping, etc.
  - Horwitz, Reps, & Binkley, TOPLAS 90
  - Reps, Horwitz, Sagiv, & Rosay, FSE 94
  - Reps & Rosay, FSE 95
- Dataflow analysis
  - Reps, Horwitz, & Sagiv, POPL 95
  - Horwitz, Reps, & Sagiv, FSE 95