Program Slicing

- Static slicing
  - Following control and data dependences
  - Executable slices
  - Intra- versus interprocedural slicing
  - Weiser algm
  - Horwitz, Reps, Binkley algm
- An "SDG" for OO programs

F. Tip, "A Survey of Program Slicing Techniques, Jl of Programming Languages, Vol 3, 1995, pp 121-189.

Slicing

- A program *slice* is comprised of the parts of a program that directly or indirectly affect a computation of interest
  - Identify a variable at a program point whose value is the focus -- *slicing criterion*
  - Introduced by Mark Weiser of Xerox Parc in late 1970s-early 1980s
    - *"A program slice S is the reduced, executable program obtained from a program P by removing statements, such that S replicates part of the behavior of P" (Tip, JPL'95)*
- *Static slice* - makes no assumptions about program input
- *Dynamic slice* - relies on a specific program run

Backward Static Slicing

Tip, JPL'95

```
read(n);
k := 1;
sum := 0;
product := 1;
while k<= n do
{   sum := sum + k;
    product := product * k;
    k := k + 1;
}
write (sum);
write (product);
```

Original Program

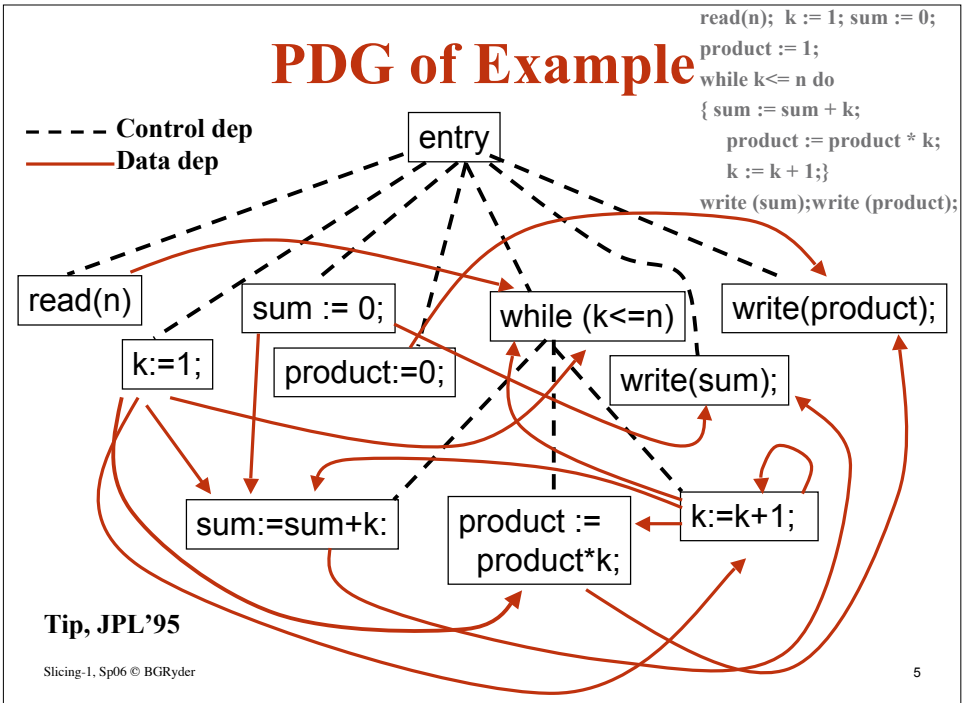
```
read (n);
k := 1;

product := 1;
while k <= n do
{
   product := product * k;
   k := k + 1;
}

write (product);//**
```

*Static slice wrt (**, product)*

Program Dependence Graph

- A data structure that removes unnecessary sequential flow of control from a program
  - Nodes are computations (e.g., statements)
  - Edges connect computations along immediate def-use dependences and along immediate control dependences
  - Historically was used for parallelization but also uncovered relevant relations to slicing

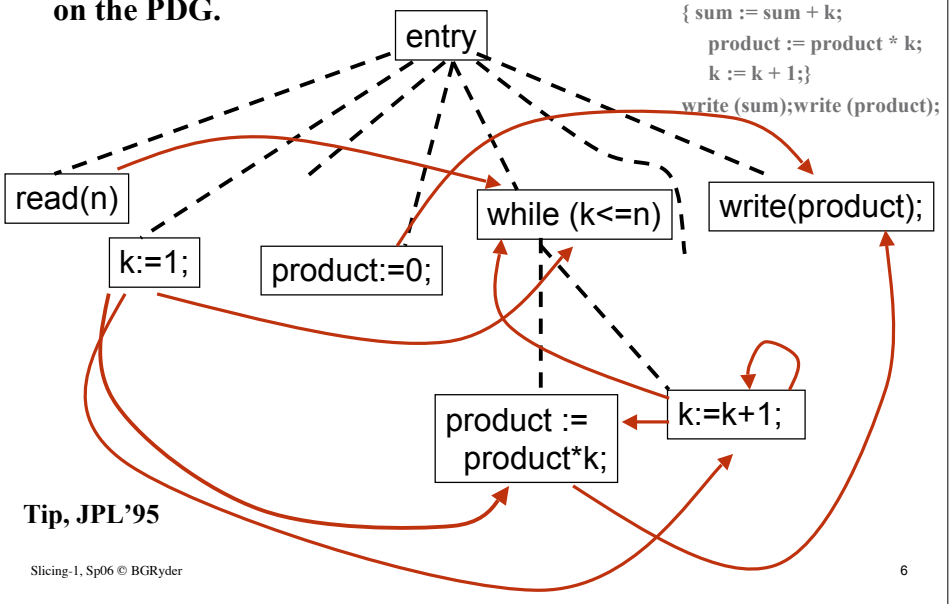
PDG of Example




```
read(n);  k := 1; sum := 0;
product := 1;
while k<= n do
{ sum := sum + k;
   product := product * k;
   k := k + 1;}
write (sum);write (product);
```

Tip, JPL'95

Same slice shown on the PDG.




```
read(n);  k := 1; sum := 0;
product := 1;
while k<= n do
{ sum := sum + k;
   product := product * k;
   k := k + 1;}
write (sum);write (product);
```

Tip, JPL'95

Mark Weiser's Static Slices

- Slicing criterion (n,V) st n is node in cfg and V is a subset of the program's variables
- Slice S will be subset of the program's statements s.t.
  - S is a valid program
  - Whenever P *halts* on an input then S also *halts* for that input, computing the *same values* for variables in V whenever the statement in node n is executed.
- Finding *minimal-sized* executable slices is *undecidable*
- Not all static slicing methods produce executable slices

How to deal with procedures?

- Weiser's slicing:
  - Calculate interprocedural Mod(P), Use(P)
  - Calculate intraprocedural slice, using Mod(P) and Use(P) at call statements
  - Iteratively generates a new slicing criterion wrt the intraprocedural slices in 2nd step to explore cfg's of affected procedures (callers and callees of procedure in focus)
    - Q called by P, slice for relevant variables (actuals substituted for formals) backward from last stmt of Q
- Problem: *calling context problem*-- too many infeasible call path
  - Also doesn't associate output params with specific input params

Example

Tip, JPL'95

```
{ 1. a := 17;
  2. b := 18;
  3. P(a,b,c,d);
  4. write(d);
}
 Procedure P(v, w, x, y)
{ 5.  x := v;
  6.  y := w;
}
```

Slicing criterion: (4, {d}).

```
{ 1. a := 17;
  2. b := 18;
  3. P(a,b,c,d);

}
 Procedure P(v, w, x, y)
{
  6.  y := w;
}
```

Weiser slice

Example

Tip, JPL'95

```
{ 1. a := 17;
  2. b := 18;
  3. P(a,b,c,d);
  4. write(d);
}
 Procedure P(v, w, x, y)
{ 5.  x := v;
  6.  y := w;
}
```

Slicing criterion: (4, {d}).

```
{
  2. b := 18;
  3. P(a,b,c,d);

}
 Procedure P(v, w, x, y)
{
  6.  y := w;
}
```

HRB slice

Horwitz-Reps-Binkley Slicing

Horwitz, Reps, Binkley, "Interprocedural Slicing Using Dependence Graphs", TOPLAS, Jan 1990, vol 12, no 1

- Three part algorithm
  - Form the System Dependence Graph (SDG) from the PDGs of each procedure
  - Compute interprocedural summary info, adding summary edges to SDG between input and output params
  - In 2 passes, extract interprocedural slices from an SDG
- Modeled parameter passing by value-result
- Slices computed are not necessarily executable
  - Need to deal with calls to procedures some of whose params are sliced away

SDG

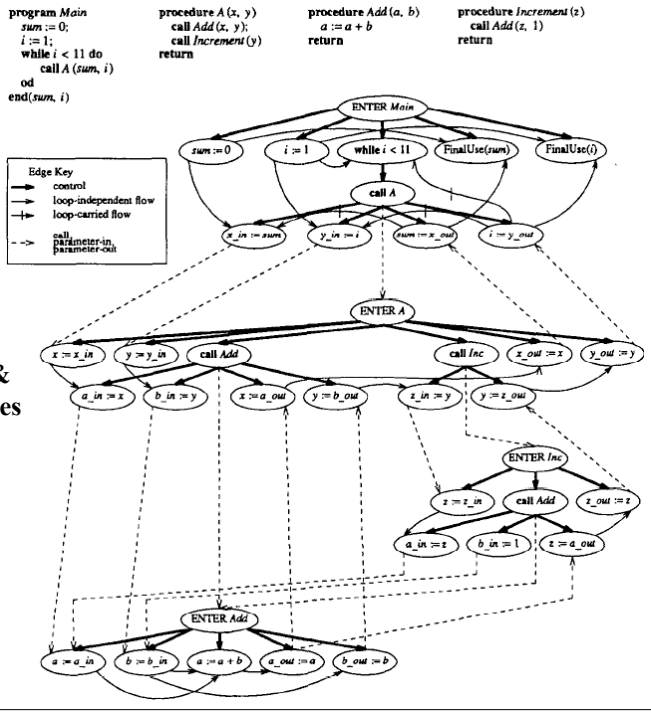
- Formed from PDGs for each procedure and *main*
  - *Intraprocedural*
    - Actual-in, actual-out vertices for params. Control dep on the call-site vertex
    - Formal-in, formal-out vertices control dep on procedure entry vertex
  - *Interprocedural*
    - Entry vertex of callee is control dep on call-site vertex
    - Param-in edge between actual-in and formal-in vertices
    - Param-out edge beetween actual-out and formal-out vertices
    - Summary edges representing transitive interprocedural data dependences;

Extracting Slices from the SDG

- **Assume start slice at vertex x**
- **Find all vertices from which x can be reached *without* descending into procedure calls**
- **Find all remaining vertices by descending into all previously encountered procedure calls,but *not ascending* up into callers.**

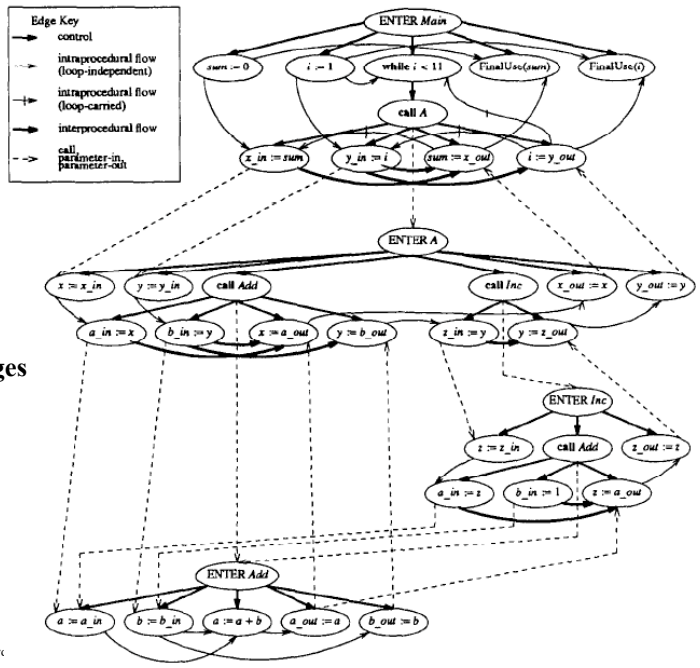
Horwitz, Reps, Binkley, TOPLAS Jan 1990

Step 1: SDG w. control dep & data-dep edges



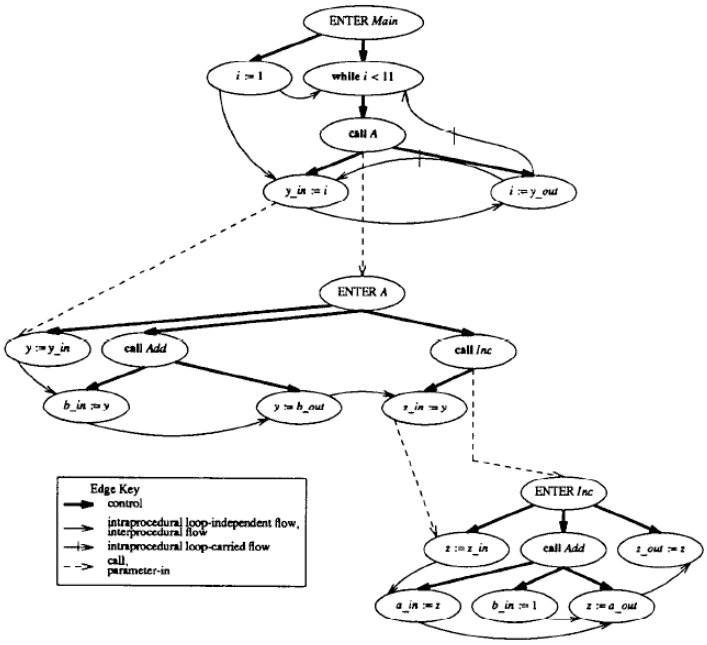
Horwitz, Reps, Binkley, TOPLAS Jan 1990

Step 2: SDG w. summary edges betw params



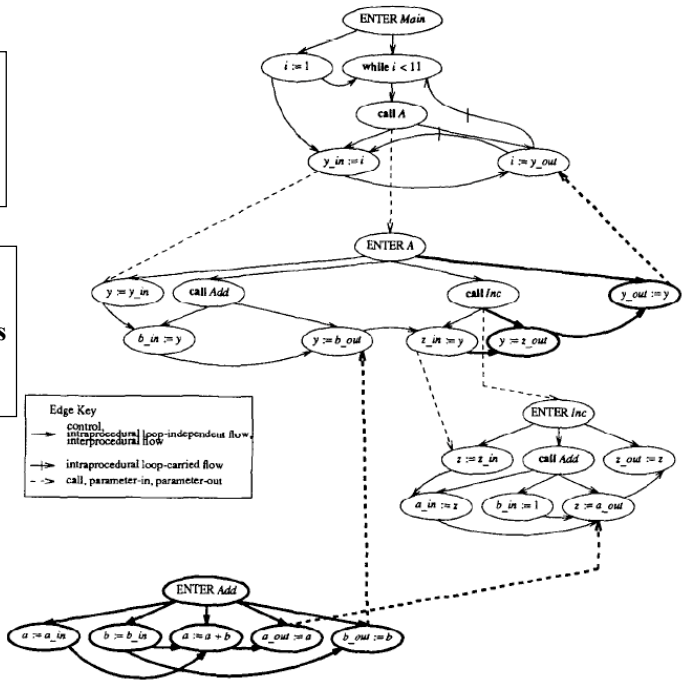
Horwitz, Reps, Binkley, TOPLAS Jan 1990

Slice wrt Inc.z(out) Phase 1



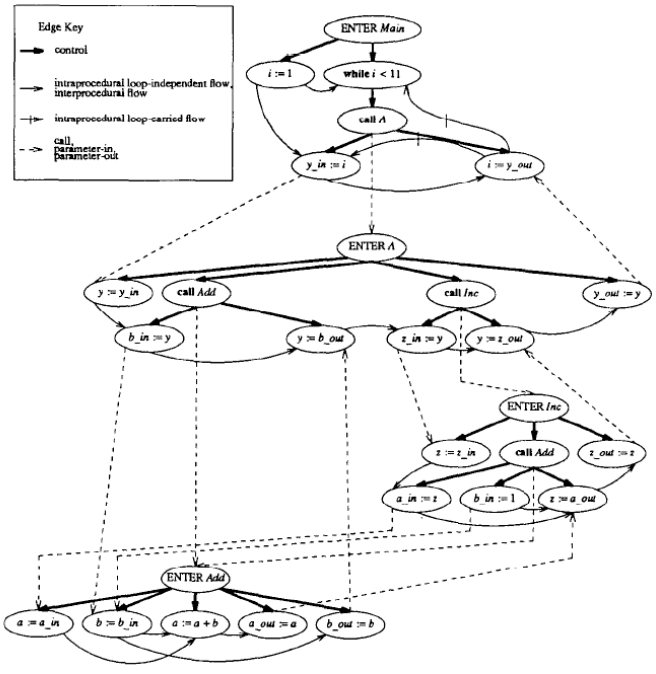
Horwitz, Reps, Binkley, TOPLAS Jan 1990

Slice wrt Inc.z(out) (new edges in bf)
Phase 2



Horwitz, Reps, Binkley, TOPLAS Jan 1990

Entire slice wrt Inc.z(out)



Imprecision

```
program Main
  sum := 0;
  i := 1;
  while i < 11 do
     call A (sum, i)
  od
end(sum, i)
```

```
procedure A (x, y)
  call Add (x, y);
  call Increment (y)
return
```

```
procedure Add (a, b)
  a := a + b
return
```

```
procedure Increment (z)
  call Add (z, 1)
return
```

Can see that Add does not affect value of b parameter, so that call of Add in A() should not be in slice of Increment wrt its output parameter z in the call in A();

Problem is that y(out) vertex is included in dependence graph of A() even though Add preserves the value of its second parameter. Can use global interprocedural mod analysis to avoid putting edges for such parameters in the graph. (Essentially don't need to copy-back these preserved-value parameters.)

ClDG: An "SDG" for OO

- **Construct SDGs called *class dependence graphs* (ClDG) for groups of OO classes (in C++)** Larsen & Harrold, ICSE'96
  - Form method PDG w entry vertex; model class w class entry vertex that connects to entries of all methods
  - Treats instance fields as globals passed into and out of every method (except constructor & destructor)
  - Derived class ClDG reuse reps of all inherited methods
  - Make explicit edges for implicit constructor calls; other method calls are analogous to in HRB SDG
  - Polymorphic call sites have new vertex representing dynamic dispatch choice w targets resolved by type-based call graph construction

ClDG construction

- Incomplete systems
  - Modeled with supernode representing a driver consisting of a nondeterministic loop choosing one of each public entry methods of class on each iteration
  - Add data dep edges for instance variables between possible method calls (var(out) to var(in))
- Case study reported
  - C++ program, 9 classes, 65 methods, hierarchy 3 levels deep; had 1257 vertices in ClDG

Problems with ClDG

- Some ClDG design choices led to imprecision
- New assumptions
  - C++ w/o exceptions
  - Points-to analysis results available
  - Data members accessible only through get methods
  - Static members are globals, static methods are global procedures
- Differences with old CLDG
  - Objects used as parameters with explicit rep of fields (truncate recursive data structures using k-limiting
  - Polymorphic objects -- a tree of possible object types (with child fields or child call-site vertices)
  - Don't always reuse method reps in derived classes (may need rep for new overloaded method or for a method that calls a redefined method in another derived class)

Liang & Harrold, "Slicing Objects Using System Dependence Graph", ICSM 1998

Slicing an Object

- **To identify slice stmts relevant to a specific object that affects the slicing criterion**
  - **Obtain the full slice**
  - **Identify method calls where object o is a receiver; this is set of *interesting* methods that captures how object o affects the slicing criterion**
  - **Identify stmts included in the slice because of these call sites -- *these are object o's slice***
  - **– Claim this is good for browsing a slice when debugging or for program understanding**