

# AdaSlicer: An Ada Program Slicer

Ricky E. Sward

Department of Computer Science  
USAF Academy, CO  
ricky.sward@usafa.edu

A.T. Chamillard

Computer Science Department  
University of Colorado Spring, CO  
chamillard@cs.uccs.edu



# Overview

- Background
- Program slicing
- Processing statements
- Inter-procedural slicing
- ASIS
- Conclusions



# Background

- Program slicing projects behavior
  - Extracts functionality from original program required to produce the value of a single variable
- Used in software maintenance
- Used in software testing for test case generation
- Used in debugging to focus the search for bugs
- Used in re-engineering



# Overview

- Background
- **Program slicing**
- Processing statements
- Inter-procedural slicing
- ASIS
- Conclusions



# Program Slicing

- A *program slice* is a projection of behavior from the original procedure
- Build slice by extracting statements needed to produce *slice variable*

```
procedure Assignscalar is
```

```
  A : Integer := 0;
```

```
  B : Integer := 1;
```

```
  C : Integer := 2;
```

```
  D : Integer := 3;
```

```
begin
```

```
  C := A + B;
```

```
  D := B * 2;
```

```
end Assignscalar;
```



```
procedure Assignscalar_D is
```

```
  B : Integer := 1;
```

```
  D : Integer := 3;
```

```
begin
```

```
  D := B * 2;
```

```
end Assignscalar_D;
```

# Program Slicing

- Static analysis tool, uses statement information
- A variable is *defined* in a statement if it can assigned a new value,
- Any variables that appear in a statement are *referenced* in that statement
- For each statement collect DEF and REF set
- Conservative slices, not necessarily minimal
- Relevant set tracks variables needed for slice



# Program Slicing

```
procedure Assignscalar is
```

```
  A : Integer := 0;
```

```
  B : Integer := 1;
```

```
  C : Integer := 2;
```

```
  D : Integer := 3;
```

```
begin
```

```
  C := A + B;
```

```
  D := B * 2;
```

```
end Assignscalar;
```



```
procedure Assignscalar_D is
```

```
  B : Integer := 1;
```

```
  D : Integer := 3;
```

```
begin
```

```
  D := B * 2;
```

```
end Assignscalar_D;
```

Relevant set: { B, D }

REF set: { B }

DEF set: { D }



# Overview

- Background
- Program slicing
- **Processing statements**
- Inter-procedural slicing
- ASIS
- Conclusions



# Processing Statements

- Intra-procedural slicing
- Begin processing with last statement
- Assignment statements

Relevant set: { E, B, D }

```
procedure Assignscalar is
  A : Integer := 0;
  B : Integer := 1;
  C : Integer := 2;
  D : Integer := 3;
  E : Integer := 4;
begin
  C := A + B;      -- process third
  D := B * 2;     -- process second
  E := D * 3;     -- process first
end Assignscalar;
```



```
procedure Assignscalar_E is
  B : Integer := 1;
  D : Integer := 3;
  E : Integer := 4;
begin
  D := B * 2;
  E := D * 3;
end Assignscalar_E;
```

# Processing Statements

- If-then-else statements
- Statements in paths are processed recursively
- Conservative approach, expression copied verbatim

```
procedure Ifthenelsescalar is
```

```
  C : Integer;
```

```
  D : Integer;
```

```
  E : Integer;
```

```
begin
```

```
  C := 0;
```

```
  D := 0;
```

```
  E := 0;
```

```
  if C = 2 then
```

```
    D := D + 1;
```

```
  else
```

```
    E := E + 1;
```

```
  end if;
```

```
end Ifthenelsescalar;
```

```
Relevant set: { C, E }
```



```
procedure Ifthenelsescalar_E is
```

```
  C : Integer;
```

```
  E : Integer;
```

```
begin
```

```
  C := 0;
```

```
  E := 0;
```

```
  if C = 2 then
```

```
    null;
```

```
  else
```

```
    E := E + 1;
```

```
  end if;
```

```
end Ifthenelsescalar_E;
```

# Processing Statements

- If-then-elsif-else statements
- Include only the elsif paths that are needed

```
procedure Elsiftest is
```

```
  F : Integer := 0;
```

```
  G : Integer := 0;
```

```
  H : Integer := 0;
```

```
  I : Integer := 0;
```

```
  J : Integer := 0;
```

```
  W : Integer := 0;
```

```
  Z : Integer := 0;
```

```
begin
```

```
  W := W - 5;
```

```
  if F = 3 then
```

```
    G := G + 1;
```

```
  elsif H = 4 then
```

```
    I := I + 2;
```

```
  elsif W = 1 then
```

```
    Z := -3;
```

```
  else
```

```
    J := J + 3;
```

```
  end if;
```

```
  W := W + 1;
```

```
end Elsiftest;
```

Relevant set: { F, H, I }



```
procedure Elsiftest_I is
```

```
  F : Integer := 0;
```

```
  H : Integer := 0;
```

```
  I : Integer := 0;
```

```
begin
```

```
  if F = 3 then
```

```
    null;
```

```
  elsif H = 4 then
```

```
    I := I + 2;
```

```
  end if;
```

```
end Elsiftest_I;
```

# Processing Statements

- Case statements
- Must include all paths

```
procedure Casescalar is
```

```
  K : Integer := 0;
```

```
  L : Integer := 0;
```

```
  O : Integer := 0;
```

```
  Q : Integer := 0;
```

```
begin
```

```
  case K is
```

```
    when 5 =>
```

```
      L := L + 1;
```

```
    when 6 =>
```

```
      O := O + 1;
```

```
    when 7 =>
```

```
      Q := Q + 1;
```

```
    when others =>
```

```
      Q := Q + 2;
```

```
  end case;
```

```
end Casescalar;
```



```
procedure Casescalar_Q is
```

```
  K : Integer := 0;
```

```
  Q : Integer := 0;
```

```
begin
```

```
  case K is
```

```
    when 5 =>
```

```
      null;
```

```
    when 6 =>
```

```
      null;
```

```
    when 7 =>
```

```
      Q := Q + 1;
```

```
    when others =>
```

```
      Q := Q + 2;
```

```
  end case;
```

```
end Casescalar_Q;
```

Relevant set: { K, Q }

# Processing Statements

```
procedure Looptest is
```

```
  C : Integer := 0;
```

```
  D : Integer := 0;
```

```
  E : Integer := 0;
```

```
begin
```

```
  C := 0;
```

```
  while C <= 10 loop
```

```
    D := D + E;
```

```
    E := E + C;
```

```
    C := C + 1;
```

```
  end loop;
```

```
end Looptest;
```

- Loop statements
- Iterate over statements until relevant set becomes stable

```
procedure Looptest_D is
```

```
  C : Integer := 0;
```

```
  D : Integer := 0;
```

```
  E : Integer := 0;
```

```
begin
```

```
  C := 0;
```

```
  while C <= 10 loop
```

```
    D := D + E;
```

```
    E := E + C;
```

```
    C := C + 1;
```

```
  end loop;
```

```
end Looptest_D;
```

```
procedure Looptest_C is
```

```
  C : Integer := 0;
```

```
begin
```

```
  C := 0;
```

```
  while C <= 10 loop
```

```
    C := C + 1;
```

```
  end loop;
```

```
end Looptest_C;
```

# Overview

- Background
- Program slicing
- Processing statements
- **Inter-procedural slicing**
- ASIS
- Conclusions



# Inter-procedural Slicing

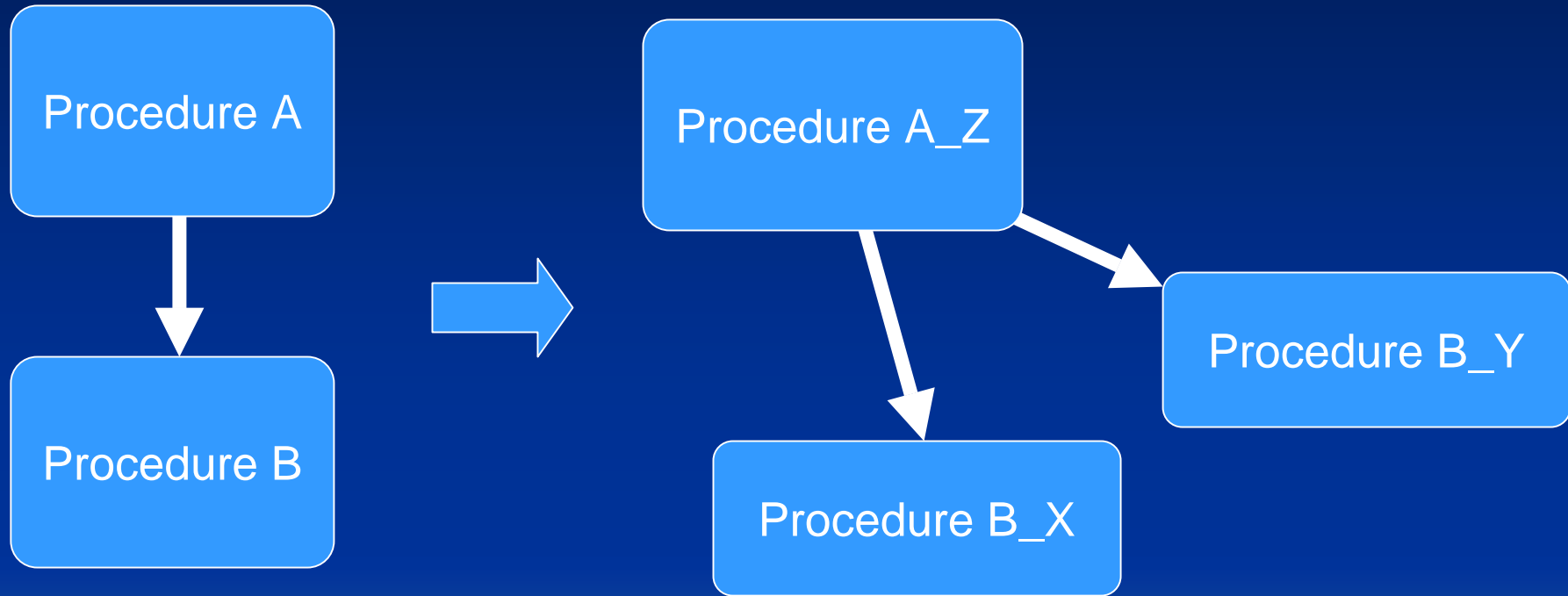
- So far dealt with slicing within a procedure
  - Intra-procedural slicing
- What about procedure call statements?
- How to handle?
- Have to slice called procedure if a variable in the relevant set is an out parameter of the called procedure



# Inter-procedural Slicing



# Inter-procedural Slicing



# Overview

- Background
- Program slicing
- Processing statements
- Inter-procedural slicing
- **ASIS**
- Conclusions



# Ada Semantic Interface Specification (ASIS)

- Procedures for accessing Ada program structure
- Used ASIS 3.15a1 & GNAT Ada Compiler 3.15a1
- Reasonable learning curve
- Examples provided with ASIS are great
- Very powerful tool



# Overview

- Background
- Program slicing
- Processing statements
- Inter-procedural slicing
- ASIS
- **Conclusions**



# Conclusions

- Currently working on graphical user interface
- Targeted at re-engineering efforts
- Experiment in under-graduate and graduate software engineering course
- Also applications in driving refactoring decisions

