

Slide Index
NPIC & HMIT (May 8, 1996):

UNIT 1 Unravel	1.0
Unravel	1.1
Talk Outline	1.2
Problems in Software Safety Evaluation	1.3
Program Slicing	1.4
Combining Slices	1.5
Programmer's View After a Slice	1.6
Computing a Program Slice	1.7
Slices on Output 1 and Output 2	1.8
Combined Slices on Output 1 and Output 2	1.9
Using Unravel on Sample Code	1.10
Unravel Performance	1.11

12 slides
April 11, 1996

UNIT 1

Unravel

Unravel

USING A PROGRAM SLICING CASE TOOL FOR EVALUATING HIGH INTEGRITY SOFTWARE SYSTEMS

Leo Beltracchi LXB@NRC.GOV

James R. Lyle JLYLE@NIST.GOV

Dolores R. Wallace DWALLACE@NIST.GOV

Nuclear Plant Instrumentation, Control, and
Human-Machine Interface Technologies
May 6-9, 1996 at The Nittany Lion Inn,
The Pennsylvania State University

Version 1.5

NPIC & HMIT (May 8, 1996): Unravel (Created April 11, 1996)

1.1

Talk Outline

- Problem Introduction
- Program Slicing
- Unravel Design
- Results Using Unravel
- Unravel Demonstration
- Functional Analysis
- Testing the Slice
- Conclusions

Problems in Software Safety Evaluation

- Need to find software faults (bugs) quickly.
- Need to identify existence of common code.
- Need to identify impact of a code change.
- Need to connect inputs (e.g. a sensor signal) to outputs (e.g. trip signal).
- Need to unravel code to confirm that different algorithms achieve the same function.
- All of the above are variations on the theme:

Need to understand an existing program.

Program Slicing

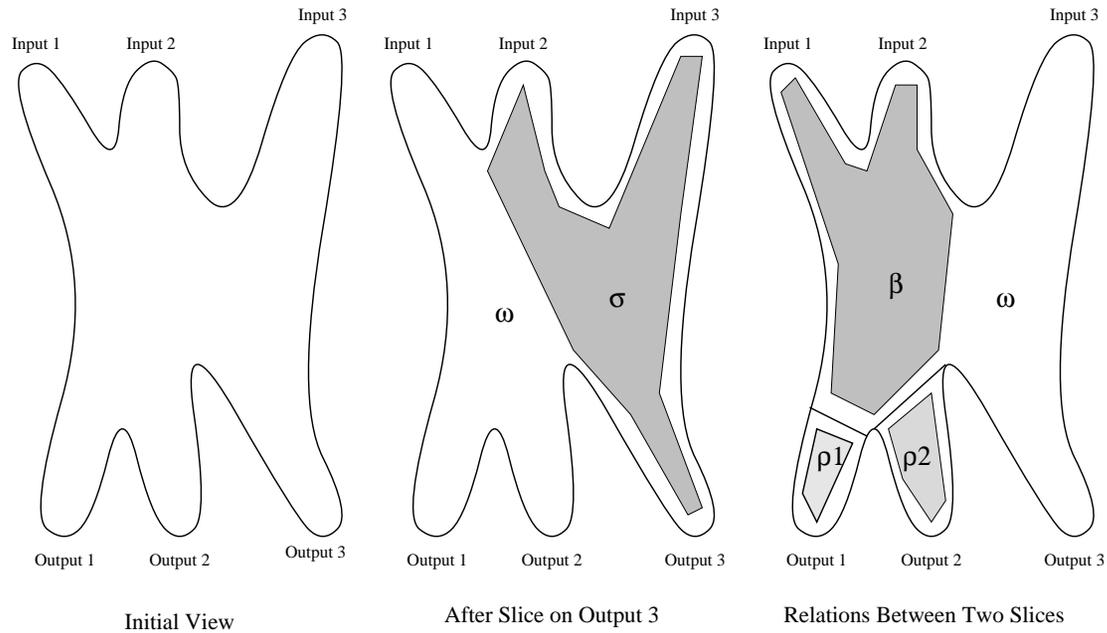
- **Program Slice:** All program statements relevant to a given computation.
- **Slicing Criterion:** Specifies the slice (computation) for a variable, \mathbf{v} , at a statement, n .
- Program slices for a given slicing criterion are obtained from a given program, \mathbf{P} , by deleting zero or more statements from \mathbf{P} , but still computing the same value for \mathbf{v} at statement n .
- Data-flow analysis is used to identify statements that may be deleted without affecting the computation.

Combining Slices

Program slices can be combined to find or exclude common code.

- The intersection of two slices (**backbone slice**) is the set of statements common to the two computations.
- A slice minus a backbone slice gives the statements unique to a computation (**program dice**).

Programmer's View After a Slice



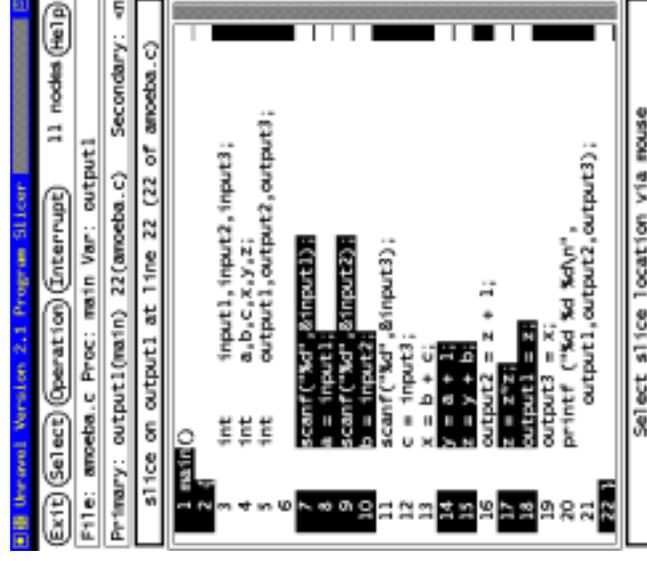
Computing a Program Slice

To locate the statements that influence variable v just before execution reaches statement m we would compute a program slice for the criterion $\langle m, v \rangle$.

1. Start with program flow-graph
2. Annotate with variables referenced and assigned to
3. the $\text{defs}(n)$ set and the slicing criterion determine inclusion
4. the $\text{refs}(n)$ set gives new criterion

$$S_{\langle m, v \rangle} = \begin{cases} S_{\langle n, v \rangle} & \text{if } v \notin \text{defs}(n) \\ \{n\} \cup S_{\langle n, x \rangle} \forall x \in \text{refs}(n) & \text{otherwise} \end{cases}$$

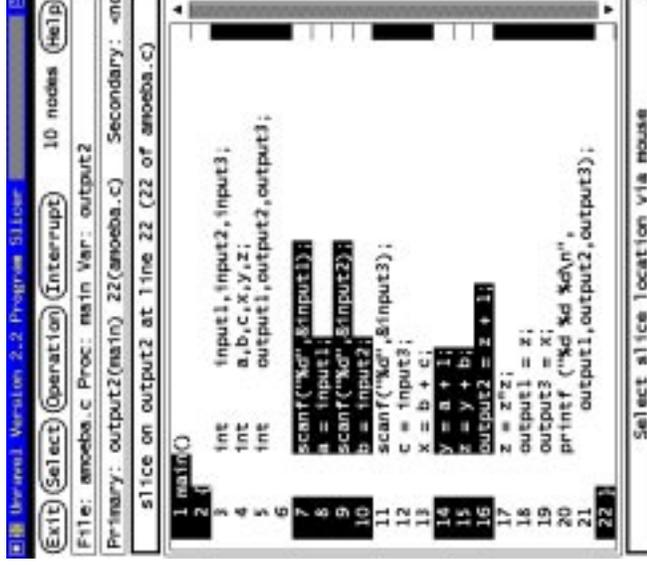
Slices on Output 1 and Output 2



Unravel Version 2.3. Program: Slice
File: amoeba.c Proc: main Var: output1 11 nodes (Help)
Primary: output1(main) 22(amoeba.c) Secondary: <n
slice on output1 at line 22 (22 of amoeba.c)

```
1 main()
2 {
3     int  input1,input2,input3;
4     int  a,b,c,x,y,z;
5     int  output1,output2,output3;
6
7     scanf("%d",&input1);
8     a = input1;
9     scanf("%d",&input2);
10    b = input2;
11    scanf("%d",&input3);
12    c = input3;
13    x = b + c;
14    y = a + 1;
15    z = y + b;
16    output2 = z + 1;
17    z = z*z;
18    output1 = z;
19    printf ("%d %d %d\n",
20            output1,output2,output3);
21
22 }
```

Select slice location via mouse



Unravel Version 2.2. Program: Slice
File: amoeba.c Proc: main Var: output2 10 nodes (Help)
Primary: output2(main) 22(amoeba.c) Secondary: <end
slice on output2 at line 22 (22 of amoeba.c)

```
1 main()
2 {
3     int  input1,input2,input3;
4     int  a,b,c,x,y,z;
5     int  output1,output2,output3;
6
7     scanf("%d",&input1);
8     a = input1;
9     scanf("%d",&input2);
10    b = input2;
11    scanf("%d",&input3);
12    c = input3;
13    x = b + c;
14    y = a + 1;
15    z = y + b;
16    output2 = z + 1;
17    z = z*z;
18    output1 = x;
19    output3 = x;
20    printf ("%d %d %d\n",
21            output1,output2,output3);
22 }
```

Select slice location via mouse

Combined Slices on Output 1 and Output 2

Unravel Version 2.2 Program Slicer 10 nodes (Help)

File: amoeba.c Proc: main Var: output1

Primary: output1(main) 22(amoeba.c) Secondary: out

Intersection of output1(main) 22(amoeba.c) & output1

```
1 main()
2 {
3     int  input1,input2,input3;
4     int  a,b,c,x,y,z;
5     int  output1,output2,output3;
6
7     scanf("%d",&input1);
8     a = input1;
9     scanf("%d",&input2);
10    b = input2;
11    scanf("%d",&input3);
12    c = input3;
13    x = b + c;
14    y = a + 1;
15    z = y + b;
16    output2 = z + 1;
17    z = x*z;
18    output1 = x;
19    output3 = x;
20    printf ("%d %d %d\n",
21           output1,output2,output3);
22 }
```

Select slice location via mouse

Unravel Version 2.3 Program Slicer 10 nodes (Help)

File: amoeba.c Proc: main Var: output3

Primary: output1(main) 22(amoeba.c) Secondary: ou

output1(main) 22(amoeba.c) sliced by output2(main)

```
1 main()
2 {
3     int  input1,input2,input3;
4     int  a,b,c,x,y,z;
5     int  output1,output2,output3;
6
7     scanf("%d",&input1);
8     a = input1;
9     scanf("%d",&input2);
10    b = input2;
11    scanf("%d",&input3);
12    c = input3;
13    x = b + c;
14    y = a + 1;
15    z = y + b;
16    output2 = z + 1;
17    z = x*z;
18    output1 = z;
19    output3 = x;
20    printf ("%d %d %d\n",
21           output1,output2,output3);
22 }
```

Select slice location via mouse

Using Unravel on Sample Code

Unravel was tried on sample safety system code, generated by an NRC staff person.

- Allowed auditor to extract a computation for manual examination
- Found questionable sharing of code

Unravel Performance

- (1) **Unravel** significantly enhances ability to analyze code.
- (2) **Unravel** is easy to operate.
- (3) **Unravel** can disclose subtle relationships in code that would require a C expert to discover.
- (4) The majority of the slices were less than 25 percent of the size of the original program (90 percent for the safety system example).
- (5) Requested slices were computed in less than one minute.

