Slide Index

NIST CSL

19 slides

UNIT 1

Program Slicing in the Presence of Pointers – SERF 93

Program Slicing in the Presence of Pointers

James R. Lyle
David Binkley

U. S. Department of Commerce
Technology Administration
National Institute of Standards and Technology
Gaithersburg, Maryland

Overview

- Slicing Research Goals
- Definitions
- An Ambiguity?
- Weiser's Algorithm Recast
- Slicing Indirect References
- Slicing Indirect Assignments
- Observations About Pointer Usage
- Pointer State Function
- Summary

Slicing Research Goals

- To compute slices faster
- To compute smaller slices
- To develop new applications
- To handle more language constructs

Definitions

Slicing Criterion $< v, n >$ **:** Specifies a slice as a variable and a location

Program Slice: All program statements relevant to a computation specified by a slicing criterion.

For a program, P, and a slicing criterion:
A program slice is obtained by deleting zero or more statements from **P**, such that **P** and the slice both compute the same value for **v** at statement n.

```
typedef struct List_struct List;
struct List_struct {
    int value;
    List    *next;
};
main()
{
    List    *at,*head = NULL,*node;
    int     n;
    n = 0;
    while (n < 10){
        node = (List *) malloc (sizeof(List));
        node -> value = n++;
        node -> next = head;
        head = node;
    }
    at = head;
    at -> value = 47;
    at = at -> next;
    at -> value = 8;
    printf ("%d\n",at -> value);
}
```

An Ambiguity?

```
main()
{
    int a,b,u,s,x;
    int *w,*y,**z;

    s = 1;
    a = 2;
    b = 3;
    scanf ("%d %d",&x,&u);

    if (x) y = &a;
    else y = &b;

    if (u) z = &y;
    else z = &w;

    w = &s;
    **z = 4;

    printf ("a %d  b %d  s %d\n",a,b,s);
}
```

Another Ambiguity?

Pointer State

Keeping Slices Executable

- Lines 11 and 12 are not relevant to the computation of s
- Since w contains the address of s, line 18 is relevant to the slice only when the condition $z = \&w$ is true.

- Two possible solutions for producing executable slices:
  1. Add statements to the slice that are irrelevant to the slicing criterion but are required to keep the slice executable.
  2. Add conditions to guard against executing statements that would cause a run-time error when the slice is executed.

Weiser's Algorithm Recast

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

1. Start with program flow-graph
2. Annotate with variables referenced and assigned to
3. the *defs(n)* set and the slicing criterion determine inclusion
4. the *refs(n)* set gives new criterion
5. $\forall n \ni m \in \mathrm{succ}(n)$

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

Compound Control

- A compound control statement has a condition controlling the execution of another statement
- **if, switch, while, for** and **do** ... **while** should be included in a program slice whenever any statement governed by the control statement is included in a slice.
- For each statement, n, associate a set, *requires(n)*, of statements that must be included in a slice containing statement n.
- The slicing rule for $v \in \text{defs(n)}$ becomes:

$$\begin{aligned} S_{<m,v>} &= \{n\} \cup S_{<n,x>} \forall x \in \text{refs(n)} \cup S_{<k,y>} \\ &\quad \forall y \in \text{refs(k)} \forall k \in \text{requires(n)} \end{aligned}$$

Indirect References

- Consider statement n: A = B + *C + **D
- defs(n) = {A} and refs(n) = {B, C, D}
- Define *iref(n)*: set of ordered pairs (var,level) indicating variable and level of indirection.
- irefs(n) = { (C,1), (D,2) }
- Define idefs similarly

Slicing Indirect References

Consider statement n: A = *B + C
Where m is a successor of n
What is $S_{<m,A>}$:

- Statement n
- $S_{<n,v>} \forall v \in \text{refs(n)}$
- $S_{<n,v>} \forall v$ that B might point at

Pointer State Function

- $P_1(n, x)$ is the set of variables that x might point to at statement n
- Note that $P_0(n, x) = \{x\}$
- $P_i(n, v) = \{x | x \in P_1(n, y) \forall y \in P_{i-1}(n, v)\}$
- For slicing arbitrary indirect references:

$$S_{<n,v>} \forall v \in P_i(n, b) \forall i, 1 \leq i \leq k, \mathrm{for}(b, k) \in \mathrm{irefs(n)}$$

Indirect Assignments

Consider n: *B = A;

Criterion $S_{<m,v>}$

Include statement n if v $\in P_1(n, B)$

- $S_{<n,v>} \forall v \in$ refs(n)
- $S_{<n,B>}$
- if size of $P_1(n, B) > 1$ then $S_{<n,v>}$

Observations About Pointer Usage

- A pointer dynamically created instances of a structure usually does not point to other structure types
- Statements change the pointer state by:
  1. allocating dynamic storage
  2. taking an address
  3. assigning one pointer variable to another
- Pointer state changes are sparsely distributed
- Pointer variables usually point to a small set of possible addresses.

Pointer State Subgraph

- A subgraph of pointer operation is embedded within a program flow-graph
- To capture the pointer state ($P_i(n, v)$) only address creation and assignment need to be tracked.
- Irrelevant control flow can be discarded
- Result is smaller flow-graph for analysis of pointer state.

Capturing Pointer State

1. Construct PSS
2. Scan PSS for address creation
3. Propagate created addresses
4. Repeat until no more changes

Summary

- An approach to slicing programs with pointers Pointer state can be used for slicing code with indirect references and assignments (more language constructs)
- PSS is significantly smaller than a program flow-graph (can compute faster with smaller graph)
- Pointers introduce some ambiguities that need examination (has size implications)