CS510 Software Engineering

Program Slicing

Asst. Prof. Mathias Payer

Department of Computer Science
Purdue University

TA: Scott A. Carr
Slides inspired by Xiangyu Zhang

http://nebelwelt.net/teaching/15-CS510-SE

Spring 2015

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

What is slicing?

Program slicing[1] is a method used by experienced computer programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a slice, is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior.

Slicing

The slice of variable v at statement S is the set of statements involved in computing v's value at S.

[1] Mark Weiser, Program slicing, ICSE'81

Slicing Example

```
1  #define N 256
2
3  void main() {
4    int i = 0;
5    int sum = 0;
6    while (i < N) {
7      sum = sum + i;
8      i = i + 1;
9    }
10   printf("Sum = %d\n", sum);
11   printf("i = %d\n", i);
12 }
```

Table of Contents

1 What is slicing?

2 Using Slicing

3 Slice Construction

4 Offline Slicing Algorithms

5 Online Slicing Algorithms

6 Forward Dynamic Slice Computation

7 Slicing Outlook

8 Summary

Use-cases for slicing

Debugging minimize program to capture essentials.

Data-Flow Testing reduce cost of regression testing after changes.

Code Reuse extract modules for reuse.

Version Integration safely integrate non-interfering extensions.

Partial Execution Replay repeat only failure-relevant part.

Partial Roll Back partially roll back a transaction.

Information Flow prevent confidential data from interacting with untrusted data.

Others...

Table of Contents

1 What is slicing?

2 Using Slicing

3 Slice Construction

4 Offline Slicing Algorithms

5 Online Slicing Algorithms

6 Forward Dynamic Slice Computation

7 Slicing Outlook

8 Summary

Slice Construction

```
1  #define N 256
2
3  void main() {
4     int i = 0;
5     int sum = 0;
6     while (i < N) {
7        sum = sum + i;
8        i = i + 1;
9     }
10    printf("Sum = %d\n", sum);
11    printf("i = %d\n", i);
12 }
```

- How can we construct a slice for # 11?
- Use PDG: Data Dependence.
- Use PGD: Control Dependence.
- Recursively enumerate all dependent statements.

Static vs. Dynamic Slicing

- Static slicing has lower overhead while dynamic slicing may use runtime information.
- Static runs into (code/data pointer) aliasing problems while dynamic runs into coverage problems.
- Static slicing may run into state explosion.
- Static slices are usually larger than dynamic slices.
- Combined static/dynamic approach allows best coverage and best results.

Dynamic Slicing

A dynamic program slice[2] is an executable subset of the original program that produces the same computations on a subset of selected variables and inputs. It differs from the static slice (Weiser, 1982, 1984) in that it is entirely defined on the basis of a computation. The two main advantages are the following: Arrays and dynamic data structures can be handled more precisely and the size of slice can be significantly reduced, leading to a finer localization of the fault.

[2] Bogdan Korel and Janusz Laski, Dynamic Program Slicing, 1988

Dynamic Slicing (2)

- The set of *executed* statements instances that contributed to the value of the criterion.
- Dynamic slicing leverages all information about a *particular* execution of a program.
- Computation through Dynamic Program Dependence Graph (DPDG): (i) each node is an executed statement; (ii) edges represent control/data dependences; dynamic slice criterion is a triplet $\{var, exec\_point, input\}$; (iii) the reachable set from a criterion constitutes a slice.
- Dynamic slices are smaller, more precise, and more helpful.

Static vs. Dynamic Slicing

- Static slicing has lower overhead while dynamic slicing may use runtime information.
- Static runs into (code/data pointer) aliasing problems while dynamic runs into coverage problems.
- Combined static/dynamic approach allows best coverage and best results.

Dynamic Slicing Example

```
1  #define N 256
2
3  void main() {
4    int i = 0;
5    int sum = 0;
6    while (i < N) {
7      sum = sum + i;
8      i = i + 1;
9    }
10   printf("Sum = %d\n", sum);
11   printf("i = %d\n", i);
12 }
```

- $slice(I@11) = \{4, 6, 8, 11\}$
- $dslice(I@11, N = 0) = \{4, 11\}$
  (11 DD 4, 11 !CD 6)
- $dslice(I@11, N = 1) = \{4, 6, 8, 11\}$
- 10': `i = 5;`
  $dslice(I@11, N = 1) = \{10', 11\}$

Control Dependence[3]

[3] B is control dependent on A iff there is a path in the CFG from A to B that does not contain the immediate dominator of A.

Dynamic Slicing Summary

- Do we still care about aliasing?
- No, a backward linear scan through the trace recovers all data dependences.
- What about control dependences?
- Still tricky. We cannot just traverse backwards to detect if a statement X is control dependent on an earlier statement, more data-structures are needed.

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

Offline Slicing

Idea: instrument the program to generate the control-flow and memory access trace, recover control dependences and data dependences offline, after trace collection.

Trace example

```
1  void main() {
2    int i = 0; trace(2, wr, &i, 0);
3    int sum = 0; trace(3, wr, &sum, 0);
4    while (i<N, trace(4, rd, &i, rd &N)) {
5      sum = sum + i; trace(5, rd, &sum, rd, &i, wr, &sum);
6      i = i + 1; trace(6, rd, &i, wr, &i);
7    }
8    printf("sum = %d\n", sum); trace(8, rd, &sum);
9    printf("i = %d\n", i); trace(9, rd, &i);
10 }
```

Offline Slicing: Data Dependence

For each "$\{R, addr\}$", traverse trace backwards to find the closest "$\{W, addr\}$", introduce a data-dependency edge. Then traverse further to find the corresponding writes of the reads on the identified write.

$\{9, rd, \&i\} \rightarrow \{6, wr, \&i\}$

$\{6, rd, \&i\} \rightarrow \{2, wr, \&i\}$

Offline Slicing: Control Dependence

Assume that there are no recursive functions and CD(i) is the set of static control dependence of i. Traverse backward in the trace and find the closest x so that x is in CD(i) then introduce a dynamic control dependence from i to x.
This does not work (well) with recursion.

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

Online Slicing Algorithms

- Offline trace traversal has huge performance implications due to frequent trace scans.
- Detect control dependence and data dependence online during execution (instead of just recording the trace).

Efficient Data Dependence Detection

- Idea: Use data structure to store last writer.
- Store last instruction that writes an address in a hashmap.
- For reads, lookup address in hashmap and add data-dependency edge.

Efficient Control Dependence Detection

Dynamic Control Dependence

y_j is dynamic control dependent (DCD) on x_i iff there exists a path from x_i to Exit that does not pass y_j and no such paths for nodes in the executed path from x_i to y_j.

We also note that all executed statements between a *predicate instance* and its *immediate post-dominator* form a **region**.

Region example

```
1 for (i=0; i<N; i++) {
2   if (i%2==0)
3     p = &a[i];
4   foo(p);
5 }
6 a=a+1;
```

```
|-  |-      1_1
|   |   |-  2_1
|   |   |-  3_1
|   |       4_1
|   |       ...
|   |   |-  1_2
|   |   |   |-2_2
|   |   |   |-3_2
|   |   |     4_2
|   |   |     ...
|   |-  |-    1_3
|-            6_1
```

Dynamic Control Dependence: Properties

1. A statement instance x_i is DCD on the predicate instance leading x_i's enclosing region.
2. Regions are disjoint or nested, they never overlap.

DCD: Property 1

A statement instance x_i is DCD on the predicate instance leading x_i's enclosing region.

Proof:

let the predicate instance be p_j and assume x_i does not DCD p_j. Therefore either (i) there is *not* a path from p_j to exit that does not pass x_i, which indicates x_i is a post-dominator of p_j, contradicting the condition that x_i is in the region delimited by p_j and its immediate post-dominator or (ii) there is a y_k in between p_j and x_i so that y_k has a path to Exit that does not pass x_i. Since p_j's immediate post-dominator is also a post-dominator of y_k, y_k and p_j's post-dominator form a smaller region that includes x_i, contradicting that p_j leads the enclosing region of x_i.

DCD: Property 2

Regions are disjoint or nested, they never overlap.
Proof: Assume there are two regions (x, y) and (m, n) that overlap. Let m reside in (x, y), thus y resides in (m, n),which implies that there is a path from m to Exit without passing y.Let the path be P. Therefore, the path from x to m and P constitute a path from x to Exit without passing y, contradicting the condition that y is a post-dominator of x.

Efficient DCD Detection

- Observation: regions have LIFO property (otherwise regions would overlap).
- Implication: the sequence of nested active regions can be maintained by a stack, called *Control Dependence Stack*.
- A region is nested in the region right below it on the stack.
- The enclosing region for the current execution point is always the top entry in the stack, therefore the execution point is control dependent on the predicate that leads the top region.
- An entry is pushed onto CDS if a branching point (predicates, switch statements, etc.) executes.
- Top is popped if the immediate post-dominator of the branching point executes, denoting the end of the current region.

Reading assignment: Bin Xin, Xiangyu Zhang, Efficient Online Detection of Dynamic Control Dependence, ISSTA'07.

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

Forward Dynamic Slice Computation

- Prior mechanisms use backward slicing: dependence graphs are traversed backwards from slicing criterion with space complexity O(execution length).
- Forward computation: a slice is represented as a set of statements that are involved in computing the value of the slicing criterion, maintain slices per variable.

Forward Dynamic Slicing

- An assignment statement execution is formulated as: $s_i : x = p_j?op(src_1, src_2, ...)$
- The statement execution instance s_i is control dependent on p_j and operates on variables of src_1, src_2, etc.
- Upon execution of s_i, the slice of x is updated to $slice(x) = s_i \cup slice(src_1) \cup slice(src_2) \cup ... \cup slice(p_j)$
- The slice of variable x is the union of the current statement, the slices of all variables that are used and the slice of the predicate instance that s_i is control dependent on. Because they are contributing to the value of x.
- Such slices are equivalent to slices computed by backwards algorithms.

Forward Dynamic Slicing (2)

- A predicate is formulated as $s_i : p_j?op(src_1, src_2, ...)$. The predicate itself is control dependent on another predicate instance p_j and the branch outcome is computed from variables of src_1, src_2, and so on.
- When executing s_i a triple of $< s_i, IPD(s), s \cup slice(src_1) \cup slice(src_2) \cup ... \cup slice(p_j) >$ is pushed to the CDS. The entry is popped at its immediate post-dominator.
- $slice(p_j)$ can be retried from the top element of the CDS.

Forward Dynamic Slicing Example

a = 1	$1_1 : a = 1$	$slice(a) = \{1\}$
b = 2	$2_1 : b = 2$	$slice(b) = \{2\}$
c = a+b	$3_1 : c = a + b$	$slice(c) = \{1, 2, 3\}$
if $a < b$ then	$4_1 : if\ \ a < b\ then$	$push(< 4_1, 6, 1, 2, 4 >)$
d = b*c	$5_1 : d = b * c$	$slice(d) = \{1, 2, 3, 4, 5\}$

Forward Dynamic Slicing (3)

- Slices are equivalent to those computed by backward slicing.
- Space complexity is bounded by $O((nr(variables) + MAX\_CDS\_DEPTH) * nr(statements))$.
- Efficiency relies on hashmap implementation and set operations.

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

Slicing Outlook

- Slicing is an approach to isolate part of the program (or execution) given a certain criterion.
- Event slicing: intrusion detection, execution fast forwarding, understanding network protocols, malware replayer.
- Forward slicing.
- Chopping.
- Probabilistic slicing.

Table of Contents

1 What is slicing?
2 Using Slicing
3 Slice Construction
4 Offline Slicing Algorithms
5 Online Slicing Algorithms
6 Forward Dynamic Slice Computation
7 Slicing Outlook
8 Summary

Summary

- Static and dynamic slicing concepts and mechanisms.
- Offline dynamic slicing algorithms based on backwards traversal of traces are inefficient.
- Online algorithms that detect data dependences and control dependences allow efficient implementation.

Questions?

?