Implementation of Interprocedural Program Slicing based on System Dependence Graph

Software Security Lab

- Introduction
- Program Slicing
- System Dependence Graph
- Data Structure of Graph
- Restore effective nodes
- Application: Flower
CHOPSTICK

- Program Slicer for C
- Based on SDG
- Forward/Backward slicing and Chopping

Program Slicing – Source

```c
void main() {
    int i = 1; int sum = 0;
    while (i<11) {
        sum = sum + i; i = i + 1;
    }
    printf("sum = %d \n", sum);
    printf("i = %d \n", i);
}
```
Program Slicing – Forward

```c
void main() {
    int i = 1; int sum = 0;
    while (i<11) {
        sum = sum + i; i = i + 1;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```

---

Program Slicing – Backward

```c
void main() {
    int i = 1; int sum = 0;
    while (i<11) {
        sum = sum + i; i = i + 1;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```
System Dependence Graph
Program Dependence Graph

Data Structure of Graph: Past

- Variants type AST
- Vertex, edge, graph

```plaintext
if(BinOp (Gt, Lval
  (Var{...}, NoOffset),
  Const (CInt64 (0L, IInt, None)),
  TInt (IInt, []),
  ...
  )
```
Data Structure of Graph : Current

- AST (vertex) ID, vertex adjacent list
- PDG graph

```
type cdsNodeType = {
  ntype: nodeType;
  mutable cfsucclist : cdsNodeListType;
  mutable cdssucclist : cdsNodeListType;
  ...
}

type cdsInfo = {
  funName :string;
  funAst :cdsNodeType IntMap.t;
  ...
}

let cds:cdsInfo list = ...
```
Data Structure of Graph: Current

- Vertex
- Multi graph
- Interprocedural vertex

Data Structure of Graph: Proposal

- SDG
- PDG
- Graph
- Edge
Data Structure of Graph: Proposal

```ocaml
module NodeMap = Map.Make(Node);
module NodeSet = Set.Make(Node);

type 'a edges = 'a NodeMap.t NodeMap.t

module Graph = sig
  val nodes: NodeSet.t
  val edgemap: bool edges
  ...
end
```

---

Restore effective nodes

- Slice node
- Function call parameter (FS)
- Criteria

- Slice node
Restore effective nodes
: Forward Slicing

- Criteria node
- Function call / parameter
- Dependence to exit node

Restore effective nodes
: Backward Slicing

- Criteria node

Application: Flower

- Functional Information Flow Graph
- Verification
Future work

- Machine code
- Framework
- Applications

Question?
Application: Flower - Verification

Control Dependent

- N® M® CFG successors®, postdominates®, not strictly postdominate.

- N® M® successors®, postdominates®, not strictly postdominate.
Slicing Algorithm: Backward

- Criteria: parameter-out dependence, call dependence, node

- node parameter-in dependence, call dependence, node

Slicing Algorithm: Forward

- Criteria: parameter-in dependence, call dependence, node

- node parameter-out dependence, call dependence, node


