Program slicing with exception handling

Carlos S. Galindo Jiménez

Universidad Politécnica de Valencia

Dec 18th, 2019
Introduction

Program slicing
The System Dependence Graph
Exception handling

Problems and proposals
Problem 1: incorrect slices with nested unconditional jumps
Problem 2: incorrect weak slices
Problem 3: incomplete catch treatment

Conclusions
Overview

Introduction
  Program slicing
  The System Dependence Graph
  Exception handling

Problems and proposals

Conclusions
Program slicing

Example adapted from [Tip95].
Program slicing

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += x;
        prod *= x;
        x --;
    }
    log("sum: " + sum);
    log("prod: " + prod);
}
```

Example adapted from [Tip95].
**Program slicing**

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += x;
        prod *= x;
        x --;
    }
    log("sum: " + sum);
    log("prod: " + prod);
}
```

Example adapted from [Tip95].
**Program slicing**

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += x;
        prod *= x;
        x--;
    }
    log("sum: " + sum);
    log("prod: " + prod);
}
```

Example adapted from [Tip95].
Program slicing

Applications

- Debugging
- Program specialization
- Software maintenance
- Code obfuscation
- Dead code removal
- Program parallelization

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += x;
        prod *= x;
        x--;
    }
    log("sum: " + sum);
    log("prod: " + prod);
}
```
**Program slicing**

**Metrics**

**Completeness.** The slice includes all instructions that are necessary.

**Correctness.** The statements included affect the slicing criterion.

Correct and complete:

Correct and incomplete:

Incorrect and complete:

Incorrect and incomplete:
**Program slicing**

**Metrics**

**Completeness.** The slice includes all instructions that are necessary.

**Correctness.** The statements included affect the slicing criterion.
**Program Slicing**

**Metrics**

**Completeness.** The slice includes all instructions that are necessary.

**Correctness.** The statements included affect the slicing criterion.

Correct and complete:

Correct and incomplete:

Incorrect and complete:

Incorrect and incomplete:
Program slicing

Metrics

Completeness. The slice includes all instructions that are necessary.

Correctness. The statements included affect the slicing criterion.
**Program slicing**

**Metrics**

**Completeness.** The slice includes all instructions that are necessary.

**Correctness.** The statements included affect the slicing criterion.

Correct and complete:

Correct and incomplete:

Incorrect and complete:

Incorrect and incomplete:
**Program slicing**

**Metrics**

**Completeness.** The slice includes all instructions that are necessary.

**Correctness.** The statements included affect the slicing criterion.
THE SYSTEM DEPENDENCE GRAPH

CREATION

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
### The System Dependence Graph

#### Creation

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x --;
    }
    log(sum);
    log(prod);
}
```

1. Enter f()
2. int sum = 0
3. int prod = 0
4. while (x > 0) {
5.    sum += sum;
6.    prod *= prod;
7.    x --;
8. }
9. log(sum);
10. log(prod);
The System Dependence Graph

Creation

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x>0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
The System Dependence Graph

Creation

```c
1 void f(int x) {
2     int sum = 0;
3     int prod = 0;
4     while (x > 0) {
5         sum += sum;
6         prod *= prod;
7         x--;
8     }
9     log(sum);
10    log(prod);
11 }
```
The System Dependence Graph

Creation

```c
void f(int x) {
  int sum = 0;
  int prod = 0;
  while (x > 0) {
    sum += sum;
    prod *= prod;
    x--;
  }
  log(sum);
  log(prod);
}
```
THE SYSTEM DEPENDENCE GRAPH

CREATION

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x --;
    }
    log(sum);
    log(prod);
}
```
THE SYSTEM DEPENDENCE GRAPH

Creation

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
THE SYSTEM DEPENDENCE GRAPH
TRAVERSAL

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x>0) {
        sum += sum;
        prod*=prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
The System Dependence Graph

Traversal

1 void f(int x) {
2     int sum = 0;
3     int prod = 0;
4     while (x>0) {
5         sum += sum;
6         prod*=prod;
7         x--;}
8     log("sum: " + sum)
9     log("prod: " + prod)
10 }

2: int sum = 0
3: int prod = 0
4: while (x > 0)
9: log("sum: " + sum)
10: log("prod: " + prod)

x = x_in
5: sum += sum
6: prod *= prod
7: x--
THE SYSTEMDEPENDENCEGRAPH

TRAVERSAL

1 void f(int x) {
2 int sum = 0;
3 int prod = 0;
4 while (x > 0) {
5 sum += sum;
6 prod *= prod;
7 x--;
8 }
9 log(sum);
10 log(prod);
11 }

The System Dependence Graph

Traversal
The System Dependence Graph

Traversal

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
The System Dependence Graph

Traversal

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x>0) {
        sum += sum;
        prod*=prod;
        x--;
    }
    log ( sum );
    log(prod);
}
```
The System Dependence Graph

Traversal

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x>0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
THE SYSTEM DEPENDENCE GRAPH

TRAVERSAL

1 void f(int x) {
2    int sum = 0;
3    int prod = 0;
4    while (x > 0) {
5       sum += sum;
6       prod *= prod;
7       x--;
8    }
9    log(sum);
10   log(prod);
11 }
The System Dependence Graph

Traversal

```c
1 void f(int x) {
2     int sum = 0;
3     int prod = 0;
4     while (x>0) {
5         sum += sum;
6         prod*=prod;
7         x--;
8     }
9     log(sum);
10     log(prod);
11 }
```
The System Dependence Graph

Traversal

```c
1 void f(int x) {
2     int sum = 0;
3     int prod = 0;
4     while (x>0) {
5         sum += sum;
6         prod*=prod;
7         x--;
8     }
9     log(sum);
10     log(prod);
11 }
```
The System Dependence Graph

Traversal

```c
void f(int x) {
    int sum = 0;
    int prod = 0;
    while (x > 0) {
        sum += sum;
        prod *= prod;
        x--;
    }
    log(sum);
    log(prod);
}
```
### Exception Handling Systems

**Java**

```java
void main () {
    try {
        f();
    } catch (Exception e) {
        log(" caught exception ");
    } catch (Throwable e) {
        log(" caught throwable ");
    }
}

void f() throws Throwable {
    switch (x) {
    case A:
        log("no error");
        break;
    case B:
        throw new Exception();
    default:
        throw new Throwable();
    }
}
```
Exception Handling Systems

Java

```java
void main() {
    try {
        f();
    } catch (Exception e) {
        log("caught exception");
    } catch (Throwable e) {
        log("caught throwable");
    }
}

void f() throws Throwable {
    switch(x) {
        case A:
            log("no error");
            break;
        case B:
            throw new Exception();
        default:
            throw new Throwable();
    }
}
```
Exception Handling Systems

Java

```java
void main () {
  try {
    f();
  } catch (Exception e) {
    log("caught exception");
  } catch (Throwable e) {
    log("caught throwable");
  }
}

void f() throws Throwable {
  switch(x) {
    case A:
      log("no error");
      break;
    case B:
      throw new Exception();
    default:
      throw new Throwable();
  }
}
```
Exception Handling Systems

Java

```java
void main() {
    try {
        f();
    } catch (Exception e) {
        log("caught exception");
    } catch (Throwable e) {
        log("caught throwable");
    }
}
```

```java
void f() throws Throwable {
    switch(x) {
    case A:
        log("no error");
        break;
    case B:
        throw new Exception();
    default:
        throw new Throwable();
    }
}
```
**Exception Handling Systems**

**Java**

```java
void main() {
    try {
        f();
    } catch (Exception e) {
        log("caught exception");
    } catch (Throwable e) {
        log("caught throwable");
    }
}

void f() throws Throwable {
    switch (x) {
    case A:
        log("no error");
        break;
    case B:
        throw new Exception();
    default:
        throw new Throwable();
    }
}
```
Exception Handling Systems

Java

```java
void main() {

    try {
        f();
    } catch (Exception e) {
        log("caught exception");
    } catch (Throwable e) {
        log("caught throwable");
    }
}

void f() throws Throwable {

    switch(x) {
    case A:
        log("no error");
        break;
    case B:
        throw new Exception();
    default:
        throw new Throwable();
    }
}
```

9/21
Exception Handling Systems
Java

- **Checked** (solid): must be caught (try-catch) or declared (f() throws T).
- **Unchecked** (dashed).
### Exception Handling Systems

#### Other programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>% usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>69.7</td>
</tr>
<tr>
<td>HTML/CSS</td>
<td>63.1</td>
</tr>
<tr>
<td>SQL</td>
<td>56.5</td>
</tr>
<tr>
<td>Python</td>
<td>39.4</td>
</tr>
<tr>
<td>Java</td>
<td>39.2</td>
</tr>
<tr>
<td>Bash/Shell/PowerShell</td>
<td>37.9</td>
</tr>
<tr>
<td>C#</td>
<td>31.9</td>
</tr>
<tr>
<td>PHP</td>
<td>25.8</td>
</tr>
<tr>
<td>TypeScript</td>
<td>23.5</td>
</tr>
<tr>
<td>C++</td>
<td>20.4</td>
</tr>
<tr>
<td>C</td>
<td>17.3</td>
</tr>
<tr>
<td>Ruby</td>
<td>8.9</td>
</tr>
<tr>
<td>Go</td>
<td>8.8</td>
</tr>
<tr>
<td>Swift</td>
<td>6.8</td>
</tr>
<tr>
<td>Kotlin</td>
<td>6.6</td>
</tr>
<tr>
<td>R</td>
<td>5.6</td>
</tr>
<tr>
<td>VBA</td>
<td>5.5</td>
</tr>
<tr>
<td>Objective-C</td>
<td>5.2</td>
</tr>
<tr>
<td>Assembly</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table: Programming language by usage in the software industry (StackOverflow’s 2019 Developer Survey)
### Exception Handling Systems

#### Other programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>% usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>69.7</td>
</tr>
<tr>
<td>HTML/CSS</td>
<td>63.1</td>
</tr>
<tr>
<td>SQL</td>
<td>56.5</td>
</tr>
<tr>
<td>Python</td>
<td>39.4</td>
</tr>
<tr>
<td>Java</td>
<td>39.2</td>
</tr>
<tr>
<td>Bash/Shell/PowerShell</td>
<td>37.9</td>
</tr>
<tr>
<td>C#</td>
<td>31.9</td>
</tr>
<tr>
<td>PHP</td>
<td>25.8</td>
</tr>
<tr>
<td>TypeScript</td>
<td>23.5</td>
</tr>
<tr>
<td>C++</td>
<td>20.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>% usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>17.3</td>
</tr>
<tr>
<td>Ruby</td>
<td>8.9</td>
</tr>
<tr>
<td>Go</td>
<td>8.8</td>
</tr>
<tr>
<td>Swift</td>
<td>6.8</td>
</tr>
<tr>
<td>Kotlin</td>
<td>6.6</td>
</tr>
<tr>
<td>R</td>
<td>5.6</td>
</tr>
<tr>
<td>VBA</td>
<td>5.5</td>
</tr>
<tr>
<td>Objective-C</td>
<td>5.2</td>
</tr>
<tr>
<td>Assembly</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table: Programming language by usage in the software industry (StackOverflow’s 2019 Developer Survey)
### Exception Handling Systems

### Other programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>% usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>69.7</td>
</tr>
<tr>
<td>HTML/CSS</td>
<td>63.1</td>
</tr>
<tr>
<td>SQL</td>
<td>56.5</td>
</tr>
<tr>
<td>Python</td>
<td>39.4</td>
</tr>
<tr>
<td>Java</td>
<td>39.2</td>
</tr>
<tr>
<td>Bash/Shell/PowerShell</td>
<td>37.9</td>
</tr>
<tr>
<td>C#</td>
<td>31.9</td>
</tr>
<tr>
<td>PHP</td>
<td>25.8</td>
</tr>
<tr>
<td>TypeScript</td>
<td>23.5</td>
</tr>
<tr>
<td>C++</td>
<td>20.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>% usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>17.3</td>
</tr>
<tr>
<td>Ruby</td>
<td>8.9</td>
</tr>
<tr>
<td>Go</td>
<td>8.8</td>
</tr>
<tr>
<td>Swift</td>
<td>6.8</td>
</tr>
<tr>
<td>Kotlin</td>
<td>6.6</td>
</tr>
<tr>
<td>R</td>
<td>5.6</td>
</tr>
<tr>
<td>VBA</td>
<td>5.5</td>
</tr>
<tr>
<td>Objective-C</td>
<td>5.2</td>
</tr>
<tr>
<td>Assembly</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table: Programming language by usage in the software industry (StackOverflow’s 2019 Developer Survey)
Overview

Introduction

Problems and proposals
- Problem 1: incorrect slices with nested unconditional jumps
- Problem 2: incorrect weak slices
- Problem 3: incomplete catch treatment

Conclusions
Problem 1
The subsumption correctness error

```java
public void f() {
    while (X) {
        if (Y) {
            if (Z) {
                A;
                break;
            }
        }
        B;
        break;
    }
    C;
}
```
**Problem 1**

The subsumption correctness error

```java
public void f() {
    while (X) {
        if (Y) {
            if (Z) {
                A;
                break;
            }
            B;
            break;
        }
        C;
    }
    D;
}
```
**Problem 1**

The subsumption correctness error

```java
public void f() {
    while (X) {
        if (Y) {
            if (Z) {
                A;
                break;
            }
            break1;
        }
        B;
        break;
    }
    C;
}
```

Diagram:

- **enter f()**
- **while (X)**
  - **if (Y)**
    - break1
    - if (Z)
      - D
      - **break2**
      - **B**
      - **C**
      - **if (Y)**
        - **if (Z)**
          - **A**
**Problem 1**

The subsumption correctness error

```java
public void f() {
    while (X) {
        if (Y) {
            if (Z) {
                A;
                break;
            }
            B;
            break;
        }
        C;
    }
    D;
}
```
Problem 1: proposed solution

The subsumption correctness error

1. Identify edges caused by unconditional jumps.

2. Remove from $E_c$ all $(a, b)$ such that $(a', b) \in E_c \land a' \in J \land (a', b), (a, b) \in E_{a'}$

$J$: unconditional jumps
$E_c$: control edges
$E_{a'}$: edges caused by $a'$
Problems and Proposals

1. Identify edges caused by unconditional jumps.

2. Remove from $E_c$ all $(a, b)$ such that $(a', b) \in E_c \land a' \in J \land (a', b), (a, b) \in E_{a'}$

$J$: unconditional jumps

$E_c$: control edges

$E_{a'}$: edges caused by $a'$
**Problem 1: proposed solution**

**The subsumption correctness error**

1. Identify edges caused by unconditional jumps.
2. Remove from $E_c$ all $(a, b)$ such that $(a', b) \in E_c \land a' \in J \land (a', b), (a, b) \in E_{a'}$

$J$: unconditional jumps
$E_c$: control edges
$E_{a'}$: edges caused by $a'$
Problem 1: proposed solution

The subsumption correctness error

1. Identify edges caused by unconditional jumps.
2. Remove from $E_c$ all $(a, b)$ such that $(a', b) \in E_c \land a' \in J \land (a', b), (a, b) \in E_{a'}$

$J$: unconditional jumps  
$E_c$: control edges  
$E_{a'}$: edges caused by $a'$
Problem 2

Unnecessary instructions in weak slicing

```c
void g() {
    int a = 1;
    while (a > 0) {
        if (a > 10)
            break;
        a++;
    }
    log(a);
}
```
**Problem 2**

**Unnecessary instructions in weak slicing**

```c
void g() {
    int a = 1;
    while (a > 0) {
        if (a > 10)
            break;
        a++;
    }
    log(a);
}
```
Problem 2

Unnecessary instructions in weak slicing

```c
void g() {
    int a = 1;
    while (a > 0) {
        if (a > 10)
            break;
        a++;
    }
    log(a);
}
```
**Problem 2: Proposed Solution**

**Unnecessary Instructions in Weak Slicing**

1. Remove all forward-jumping unconditional jump's nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).

```cpp
int a = 1
while (a > 0)
    if (a > 10)
        log(a)
    a++
    break
f()
```
Problem 2: Proposed Solution

Unnecessary Instructions in Weak Slicing

1. Remove all forward-jumping unconditional jump’s nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).
**Problem 2: Proposed Solution**

_Unnecessary Instructions in Weak Slicing_

1. Remove all forward-jumping unconditional jump’s nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).
**Problem 2: Proposed Solution**

**Unnecessary Instructions in Weak Slicing**

1. Remove all forward-jumping unconditional jump’s nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).
**Problem 2: Proposed Solution**

**Unnecessary Instructions in Weak Slicing**

1. Remove all forward-jumping unconditional jump’s nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).
Problem 2: proposed solution

Unnecessary instructions in weak slicing

1. Remove all forward-jumping unconditional jump’s nodes.
2. Traverse the graph.
3. Re-add all nodes removed if there is any statement in the slice after their destination.
4. If (3) changed the graph, goto (2).
**Problem 3**

**The Lack of Dependencies of catch Statements**

```
1 int x;
2
3 void main() throws Exception {
4    try {
5        f();
6    } catch (Exception e) {
7        log("error");
8    }
9
10    f();
11 }
12
13 void f() throws Exception {
14    x++;
15    if (x > 0)
16        throw new Exception();
17    log(x);
18 }
```
**PROBLEM 3**

**THE LACK OF Dependencies of catch Statements**

```java
1 int x;
2
3 void main() throws Exception {
4    try {
5        f();
6    } catch (Exception e) {
7        log("error");
8    }
9
10   f();
11 }
12
13 void f() throws Exception {
14    x++;
15    if (x > 0)
16        throw new Exception();
17    log(x);
18 }
```
**Problem 3**

The Lack of Dependencies of `catch` Statements

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    f();
}

void f() throws Exception {
    x ++;
    if (x > 0)
        throw new Exception();
    log(x);
}
```
**Problem 3**

**The Lack of Dependencies of catch Statements**

```java
1 int x;
2
3 void main() throws Exception {
4   try {
5     f();
6   } catch (Exception e) {
7     log("error");
8   }
9
10  f();
11 }
12
13 void f() throws Exception {
14   x ++;
15   if (x > 0)
16     throw new Exception();
17   log(x);
18 }
```
**Problem 3**

The Lack of Dependencies of `catch` Statements

```java
1 int x;
2
3 void main() throws Exception {
4    try {
5        f();
6    } catch (Exception e) {
7        log("error");
8    }
9
10   f();
11 }
12
13 void f() throws Exception {
14    x ++;
15    if (x > 0)
16        throw new Exception();
17    log(x);
18 }
```
**Problem 3**

**The Lack of Dependencies of catch Statements**

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }

    f();
}

void f() throws Exception {
    x++;  
    if (x > 0) 
        throw new Exception();
    log(x);
}
```
**Problem 3**

The Lack of Dependencies of catch Statements

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    x = 0;
    f();
}

void f() throws Exception {
    x ++;
    if (x > 0)
        throw new Exception();
    log(x);
}
```
### Problem 3

**The Lack of Dependencies of catch Statements**

```java
int x;
void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    x = 0;
    f();
}
void f() throws Exception {
    x ++;
    if (x > 0)
        throw new Exception();
    log(x);
}
```
Problem 3

The Lack of Dependencies of catch Statements

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    x = 0;
    f();
}

void f() throws Exception {
    x ++;
    if (x > 0) 
        throw new Exception();
    log(x);
}
```
**Problem 3**

The Lack of Dependencies of `catch` Statements

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    x = 0;
    f();
}

void f() throws Exception {
    x ++;
    if (x > 0)
        throw new Exception();
    log(x);
}
```
**Problem 3**

**The Lack of Dependencies of catch Statements**

```java
int x;

void main() throws Exception {
    try {
        f();
    } catch (Exception e) {
        log("error");
    }
    x = 0;
    f();
}

void f() throws Exception {
    x ++;
    if (x > 0)
        throw new Exception();
    log(x);
}
```
**Problem 3: Proposed Solution**

The lack of dependencies of catch statements
**Problem 3: Proposed Solution**

The lack of dependencies of catch statements

- **Problem 3:** proposed solution
- The lack of dependencies of catch statements
Problem 3: proposed solution

The lack of dependencies of catch statements
Problem 3: Proposed Solution

The lack of dependencies of catch statements
**Problem 3: Proposed Solution**

**The lack of dependencies of catch statements**
Overview

Introduction

Problems and proposals

Conclusions
Conclusions

- Program slicing: a powerful technique, not yet complete for commonly used programming languages.
- Efficiency vs. completeness and correctness.
- Results: 3 problems and proposed solutions.
Conclusions

- Program slicing: a powerful technique, not yet complete for commonly used programming languages.
- Efficiency vs. completeness and correctness.
- Results: 3 problems and proposed solutions.
Future work

- Improve correctness of try-catch.
- Implement into existing software tools, benchmark against state of the art.
- Adapt to other variants of program slicing.
- Redefine control dependence (extend Danicic’s work [Dan+11]), execution dependence vs. presence dependence.
Related work

Change to state of the art and move to introduction

► 1988: Horwitz, Reps and Ball [HRB88] present the SDG.
► 1993: Ball and Horwitz [BH93] present SDG with unconditional jumps
► 2003: Allen and Horwitz [AH03] improves [SH98].
► 2004: Jo and Chang [JC04] present an alternative construction of the graph
  (not demonstrated, not an improvement).
► 2006: Jiang et al. [Jia+06] propose a solution for C++, not applicable.
► 2011: Prabhy, Maeda and Blakrishnan [PMB11] propose another solution for
  C++, no notable improvement.
► 2011: Jie and Shu-juan [JSJ11] introduce Object-Oriented + Exception SDG,
  same errors as [AH03].
Related work

Change to state of the art and move to introduction

▶ 1988: Horwitz, Reps and Ball [HRB88] present the SDG.
▶ 1993: Ball and Horwitz [BH93] present SDG with unconditional jumps
▶ 2003: Allen and Horwitz [AH03] improves [SH98].
▶ 2004: Jo and Chang [JC04] present an alternative construction of the graph (not demonstrated, not an improvement).
▶ 2006: Jiang et al. [Jia+06] propose a solution for C++, not applicable.
▶ 2011: Jie and Shu-juan [JSJ11] introduce Object-Oriented + Exception SDG, same errors as [AH03].
Program slicing
Strong and Weak Slices

Definition (Strong slice [Wei81])
Given a program $P$, its slice $S$ w.r.t. a slicing criterion $SC$ is a subset of the statements ($S \subseteq P$) such that the execution of $S$ yields the same sequence of values on $SC$ as the execution of $P$.

Definition (Weak slice [BG96])
Given a program $P$, its slice $S$ w.r.t. a slicing criterion $SC$ is a subset of the statements ($S \subseteq P$) such that the execution of $S$ yields a sequence of values on $SC$ ($seq_S$), the execution of $P$ yields a sequence of values on $SC$ ($seq_P$), and $seq_P$ is a prefix of $seq_S$. 
Program Slicing

Strong and Weak Slices: Example

<table>
<thead>
<tr>
<th>Original program</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice A</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Slice B</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
</tr>
<tr>
<td>Slice C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table: Sequences of values produced on the slicing criterion.


Bibliography II


