Generating Tests from Counterexamples

Sebastian Henneberg

November 13, 2012
Motivation

- testing requires considerable amount of time
- especially if managers have specific aims (e.g. specific percentage of coverage)
- even well trained testers forget test cases
- automatic generation of test vectors
- reduce user interaction

⇒ ease development
⇒ detect problems or code smells
⇒ increase development speed
Control Flow Graph (CFG) and Control Flow Automaton (CFA)

CFG and CFA

- provides graphical view of a program
- represent all possible execution paths
Control Flow Graph (CFG) and Control Flow Automaton (CFA)

CFG and CFA

- provides graphical view of a program
- represent all possible execution paths

CFG

- nodes represent either
  - basic block or
  - branch statement
- edges represent jumps to lines

CFA

- nodes represent *locations*
- edges represent either
  - basic block or
  - branch statement
- edges are annotated with predicates

\[ a = a + 1; \]
\[ b = 1; \]
\[ c = 1; \]
\[ a > 10 \]
Example of a Control Flow Automaton (CFA)

```c
int middle(int x, int y, int z) {
    int m = z;
    if (y < z) {
        if (x < y) m = y;
        else if (x < z) m = x;
    } else {
        if (x > y) m = y;
        else if (x > z) m = x;
    }
    return m;
}
```
Example of a Control Flow Automaton (CFA)

```c
int middle(int x, int y, int z) {
    int m = z;
    if (y < z)
        if (x < y)
            m = y;
        else if (x < z)
            m = x;
    else
        if (x > y)
            m = y;
        else if (x > z)
            m = x;
    return m;
}
```
Example of a Control Flow Automaton (CFA)

1 \textbf{int} \textbf{middle}(\textbf{int} \ x, \ \textbf{int} \ y, \ \textbf{int} \ z) \{ \\
2 \quad \textbf{int} \ m = z; \\
3 \quad \textbf{if} \ (y < z) \\
4 \qquad \textbf{if} \ (x < y) \\
5 \qquad \quad m = y; \\
6 \quad \textbf{else if} \ (x < z) \\
7 \qquad m = x; \\
8 \quad \textbf{else} \\
9 \qquad \textbf{if} \ (x > y) \\
10 \qquad \quad m = y; \\
11 \quad \textbf{else if} \ (x > z) \\
12 \qquad m = x; \\
13 \quad \textbf{return} \ m; \\
14 \}

Sebastian Henneberg: Generating Tests from Counterexamples
Trace Formula (TF)

2

Block(m=z;)

3

Pred(y<z)

Pred(y≥z)

4

Pred(x≥y)

Pred(x<y)

6

Pred(x≥z)

Pred(x<z)

5

Block(m=y;)

9

Pred(x≥y)

Pred(x<y)

7

Block(m=x;)

10

Block(m=x;)

11

Pred(x>z)

Pred(x≥z)

12

Block(m=y;)

13

Pred(x≥z)

Block(m=y;)

m=z; assume(y<z) assume(x<y)
Trace Formula (TF)

location: 2

trace:

trace formula:
Trace Formula (TF)

TRACE FORMULA

```
trace formula:
m = z
```

```
location: 2
```

```
trace:
m = z;
```

Sebastian Henneberg: Generating Tests from Counterexamples 5
Trace Formula (TF)

location: 3

trace:

\[ m = z \]

trace formula:

\[ m = z \]
Trace Formula (TF)

location: 3

trace:
  m = z;
  assume(y < z)

trace formula:
  \( m = z \land y < z \)
Trace Formula (TF)

location: 4

trace:
  m=z;
  assume(y<z)

trace formula:
  \( m = z \land y < z \)
Trace Formula (TF)

location: 4
trace:
  m=z;
  assume(y<z)
  assume(x<y)
trace formula:
  \( m = z \land y < z \land x < y \)
Trace Formula (TF)

location: 5

trace:
  m=z;
  assume(y<z)
  assume(x<y)

trace formula:
\[ m = z \land y < z \land x < y \]
Trace Formula (TF)

location: 5

trace:
  m=z;
  assume(y<z)
  assume(x<y)

trace formula:
  m = z \land y < z \land x < y

x < y < z = m
Trace Formula (TF)

location: 5

trace:

- \( m = z \);
- \( \text{assume}(y < z) \);
- \( \text{assume}(x < y) \);

trace formula:

\[
\begin{align*}
    m &= z \\
    x &= 0, y = 1, z = m = 2
\end{align*}
\]
Trace Formula (TF)

location: 5

trace:
  m=z;
  assume(y<z)
  assume(x<y)

trace formula:
  \( m = z \land y < z \land x < y \)

x < y < z = m

x = 0, y = 1, z = m = 2

test vector:
  \{x = 0, y = 1, z = 2\}
Reachability

CFA is always given
(generated from program source)

red tree: partial reachability tree
red path: p-trace
i.e. location 5 is reachable
Reachability

CFA is always given
(generated from program source)

red tree: *partial reachability tree*
red path: *p-trace*
i.e. location 5 is reachable
Reachability

CFA is always given
(generated from program source)

red tree: partial reachability tree
red path: p-trace
i.e. location 5 is reachable

blue tree: complete reachability tree
i.e. location 12 is unreachable
→ location 11 unreachable
int middle(int x, int y, int z) {
    int m = z;
    if (y < z)
        if (x < y)
            m = y;
        else if (x < z)
            m = x;
    else
        if (x > y)
            m = y;
        else if (x > z)
            m = x;
    return m;
}
lvalues

- \( m = z \); is different from \( m = z \)
lvalues

- $m = z$; is different from $m = z$

- assignment vs. equality

- multiple assignments may lead to contradictions
  $a = 1; a = 2;
lvalues

- $m = z$; is different from $m = z$

- assignment vs. equality

- multiple assignments may lead to contradictions
  
  $a = 1; a = 2;$
  
  $\rightarrow a = 1 \land a = 2$
  
  $\rightarrow$ unsatisfiable
lvalues

- $m = z$; is different from $m = z$

- assignment vs. equality

- multiple assignments may lead to contradictions
  
a = 1; a = 2;
  \[ \rightarrow a = 1 \land a = 2 \]
  \[ \rightarrow \text{unsatisfiable} \]
Ivalues

- $m = z$; is different from $m = z$

- assignment vs. equality

- multiple assignments may lead to contradictions
  
  $a = 1; a = 2;$
  
  $\rightarrow a = 1 \land a = 2$
  
  $\rightarrow$ unsatisfiable

- Ivalues are used to represent values at specific locations

- the tuple $\langle l, \theta(l) \rangle$ represents the mappings to values at location $l$
Security example

- trace formula can be extended by custom predicate $p$
- allows to detect locations with respect to $p$
- very useful to assert certain properties

check reachable locations with root permissions ($p := uid = 0$)

```c
1  work_and_drop_priv() {
2     FILE *fp = fopen(FILENAME,"w");
3     if(!fp) {
4         return;
5     }
6     // work
7     seteuid(saved_uid);
8 }
```
Limitations

fully-featured C programs can not be analysed with the presented framework
Limitations

fully-featured C programs can not be analysed with the presented framework

- considered: function calls without pointers
- assumption: all variables are int’s
- conversions are possible but complex
- paper focusses on feasibility
## Experimental results

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>CFA locations</th>
<th>Locations</th>
<th>Tests</th>
<th>Predicates</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Live</td>
<td>Dead</td>
<td>Fail</td>
<td>Total</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>kbfiltr</td>
<td>5933</td>
<td>381</td>
<td>298</td>
<td>39</td>
<td>112</td>
<td>5 min</td>
</tr>
<tr>
<td>floppy</td>
<td>8570</td>
<td>1039</td>
<td>780</td>
<td>111</td>
<td>239</td>
<td>25 min</td>
</tr>
<tr>
<td>cdaudio</td>
<td>8921</td>
<td>968</td>
<td>600</td>
<td>85</td>
<td>246</td>
<td>25 min</td>
</tr>
<tr>
<td>parport</td>
<td>12288</td>
<td>2518</td>
<td>1895</td>
<td>213</td>
<td>509</td>
<td>91 min</td>
</tr>
<tr>
<td>parclass</td>
<td>30380</td>
<td>1663</td>
<td>1326</td>
<td>219</td>
<td>343</td>
<td>42 min</td>
</tr>
<tr>
<td>ping</td>
<td>1487</td>
<td>814</td>
<td>754</td>
<td>134</td>
<td>41</td>
<td>7 min</td>
</tr>
<tr>
<td>ftpd</td>
<td>8506</td>
<td>6229</td>
<td>4998</td>
<td>231</td>
<td>380</td>
<td>1 d</td>
</tr>
</tbody>
</table>
Conclusion

Test generation

▶ test vectors and drivers can be determined automatically
▶ guarantees full coverage (for live locations) and scales well

Security / Error detection with customized $p$

▶ (un)reachable locations w.r.t. $p$ can be determined
▶ allows e.g. to verify certain security requirements

Dead code detection with $p = true$

▶ better comprehension of reachability
▶ allows cleanup