Automated White-Box Testing
Beyond Branch Coverage

Sébastien Bardin, Nikolai Kosmatov, Mickaël Delahaye

CEA LIST, Software Safety Lab
(Paris-Saclay, France)
DSE is GREAT for automating structural testing
✓ very powerful approach to (white box) test generation
✓ many tools and many successful case-studies since mid 2000’s
The problem

DSE is GREAT for automating structural testing
✓ very powerful approach to (white box) test generation
✓ many tools and many successful case-studies since mid 2000’s

Yet, no real support for many structural coverage criteria
[except branch coverage]

Would be useful :
- when required to produce tests achieving some criterion
- for producing “good” tests for an external oracle
  [functional correctness, security, performance, etc.]
Our goals

- extend DSE to a large class of structural coverage criteria
  - recent efforts in this direction through instrumentation
  - but exponential explosion of the search space
- support these criteria in a unified way
- bonus: what about infeasible requirement detection?
Outline

- Introduction
- Labels
- A label-based automated testing framework [TAP 14]
- Efficient DSE for Labels [ICST 14]
- Infeasible label detection [ongoing work]
- Conclusion
A well-defined specification mechanism for coverage criteria

- based on predicates, can easily encode a large class of criteria
- in the scope of standard program analysis techniques

Given a program $P$, a label $l$ is a pair $(\text{loc}, \varphi)$, where:

- $\varphi$ is well-defined in $P$ at location $\text{loc}$
- $\varphi$ contains no side-effect expression

**Basic definitions**

- an annotated program is a pair $\langle P, L \rangle$, with $L$ set of labels
- a test datum $t$ covers $l$ if $P(t)$ reaches $\text{loc}$ and satisfies $\varphi$
Simulation of standard coverage criteria

statement_1;
if (x== y && a<b)
    {...};
statement_3;

− − − − − →

statement_1;
// l1: x==y && a<b
// l2: !(x==y && a<b)
if (x== y && a<b)
    {...};
statement_3;

Decision Coverage (DC)
Simulation of standard coverage criteria

```
statement_1;
if (x==y && a<b)
    {...};
statement_3;
```

```
statement_1;
// l1: x==y
// l2: !(x==y)
// l3: a<b
// l4: !(a<b)
if (x==y && a<b)
    {...};
statement_3;
```

Condition Coverage (**CC**)

Bardin et al. Dagstuhl seminar 14442
Simulation of standard coverage criteria

```c
statement_1;
if (x==y && a<b)
{
    ...
};
statement_3;
```

Multiple-Condition Coverage (MCC)

```c
statement_1;
// l1: x==y && a<b
// l2: x==y && a>=b
// l3: x!=y && a<b
// l4: x!=y && a>=b
if (x==y && a<b)
{
    ...
};
statement_3;
```
Simulation of standard coverage criteria

OBJ: generic specification mechanism for coverage criteria

- IC, DC, FC, CC, MCC, GACC
- large part of Weak Mutations
- Input Domain Partition
- Run-Time Error
Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion
LTTest overview [TAP 14]

Supported criteria
- DC, CC, MCC
- FC, IDC, WM

Criteria encoded with labels
- managed in a unified way
- rather easy to add new ones
Overview

Program → Annotated Program → Existing Test Suite → Test Execution → Coverage Report
DSE* procedure

- DSE with native support for labels
- extension of PATHCRAWLER
Service cooperation
- share label statuses
- Covered, Infeasible, ?
Reuse static analyzers from Frama-C
- sound detection!
- several modes: VA, WP, VA $\otimes$ WP

Service cooperation
- share label statuses
- Covered, Infeasible, ?
Reuse static analyzers from Frama-C
- sound detection!
- several modes: VA, WP, VA $\otimes$ WP

Service cooperation
- share label statuses
- Covered, Infeasible, ?
Supported criteria
- all managed in a unified way
- rather easy to add new ones

Service cooperation
- share label statuses
- Covered, Infeasible, ?

Prototype
- plugin of FRAMA-C (open-source)
- based on PATHCRAWLER for test generation
- the plugin itself is open-source except test generation
Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion
Covering label \( l \) \( \iff \) Covering branch True
Covering label $l \iff$ Covering branch True

- ✓ sound & complete instrumentation
- × dramatic overhead [theory & practice]
Direct instrumentation is not good enough

Non-tightness 1

× P’ has exponentially more paths than P

Diagram:

- Node 1
- Node p1 with paths True and False
- Node 2
- Node pN with paths True and False
- Node N
- Total paths: $2^N$
Direct instrumentation is not good enough

Non-tightness 1
- P’ has exponentially more paths than P

Non-tightness 2
- Paths in P’ too complex
  - at each label, require to cover \( p \) or to cover \( \neg p \)
  - \( \pi’ \) covers up to \( N \) labels
Our approach

The DSE* algorithm [ICST 14]

- Tight instrumentation $P^*$: totally prevents “complexification”
- Iterative Label Deletion: discards some redundant paths
- Both techniques can be implemented in black-box
DSE*: Tight Instrumentation

Covering label l ⇔ Covering exit(0)

✓ sound & complete instrumentation
✓ no complexification of the search space
Observations

- we need to cover each label only once
- yet, DSE explores paths of $P^*$ ending in already-covered labels
- burden DSE with “useless” paths w.r.t. label coverage

Solution: Iterative Label Deletion

- keep a cover status for each label
- symbolic execution ignores paths ending in covered labels
- dynamic execution updates cover status [truly requires DSE]

Iterative Label Deletion is relatively complete w.r.t. label coverage
Goal of experiments

- evaluate DSE\(^*\) versus DSE’
- evaluate overhead of handling labels

Benchmark programs

- 12 programs taken from standard benchmarks (Siemens, Verisec, MediaBench) [beware: small programs]
- 3 coverage criteria: CC, MCC, WM
  [uncoverable labels not discarded]
## Experiments (2)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>DSE</th>
<th>DSE'</th>
<th>DSE*</th>
</tr>
</thead>
<tbody>
<tr>
<td>utf8-5</td>
<td>wm 84 /</td>
<td>#paths time cover</td>
<td>680</td>
<td>11,111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2s</td>
<td>40s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82/84</td>
</tr>
<tr>
<td>utf8-7</td>
<td>wm 84 /</td>
<td>#paths time cover</td>
<td>3,069</td>
<td>81,133</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.8s</td>
<td>576s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82/84</td>
</tr>
<tr>
<td>tcas</td>
<td>wm 111 /</td>
<td>#paths time cover</td>
<td>4,420</td>
<td>300,213</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.6s</td>
<td>662s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101/111</td>
</tr>
<tr>
<td>replace</td>
<td>wm 79 /</td>
<td>#paths time cover</td>
<td>866</td>
<td>87,498</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2s</td>
<td>245s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70/79</td>
</tr>
<tr>
<td>get_tag-6</td>
<td>cc 20 /</td>
<td>#paths time cover</td>
<td>76,456</td>
<td>TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3,011s</td>
<td>1,512s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76,481</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gd-5</td>
<td>wm 47 /</td>
<td>#paths time cover</td>
<td>76,456</td>
<td>TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3,011s</td>
<td>TO</td>
</tr>
<tr>
<td>gd-6</td>
<td>wm 63 /</td>
<td>#paths time cover</td>
<td>14,516</td>
<td>TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50s</td>
<td>TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experiments (3)

- DSE’ : 4 TO, max overhead 122x [excluding TO]
- DSE* : no TO, max overhead 7x, average : 2.4x
- cherry picking : 94s vs TO [1h30]
Experiments (3)

- **DSE’**: 4 TO, max overhead 122x [excluding TO]
- **DSE***: no TO, max overhead 7x, average: 2.4x
- cherry picking: 94s vs TO [1h30]

Conclusion

- **DSE*** performs significantly better than **DSE’**
- The overhead of handling labels is kept reasonable
- still room for improvement
Experiments (3)

- DSE': 4 TO, max overhead 122x [excluding TO]
- DSE*: no TO, max overhead 7x, average: 2.4x
- cherry picking: 94s vs TO [1h30]

Conclusion

- DSE* performs significantly better than DSE'
- The overhead of handling labels is kept reasonable
- still room for improvement

- also: high coverage [min: 61%, max: 100%, mean: 91% - see after]
- also: DSE* covers more than DSE [0%-39%]
Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion
Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA $\oplus$ WP : combination
Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA ⊕ WP : combination

Detection power

<table>
<thead>
<tr>
<th></th>
<th>#Lab</th>
<th>#Inf</th>
<th>VA</th>
<th>WP</th>
<th>VA ⊕ WP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>#D</td>
<td>#D</td>
<td>#D</td>
</tr>
<tr>
<td>Total</td>
<td>1270</td>
<td>121</td>
<td>84</td>
<td>73</td>
<td>118</td>
</tr>
<tr>
<td>%D</td>
<td>69%</td>
<td>60%</td>
<td>98%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>%D</td>
<td>0%</td>
<td>0%</td>
<td>67%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>29</td>
<td>29</td>
<td>15</td>
<td>29</td>
<td>100%</td>
</tr>
<tr>
<td>%D</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>4.7</td>
<td>3.2</td>
<td>2.8</td>
<td>4.5</td>
<td>95%</td>
</tr>
<tr>
<td>%D</td>
<td>63%</td>
<td>82%</td>
<td>95%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# D : number of detected infeasible labels
% D : ratio of detected infeasible labels
(speed : ≤ 1s per label)
**Infeasibility detection** [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA ⊕ WP : combination

**Improving DSE* : reported coverage ratio**

<table>
<thead>
<tr>
<th>Detection Method</th>
<th>None</th>
<th>VA</th>
<th>WP</th>
<th>VA ⊕ WP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>90%</td>
<td>97%</td>
<td>96%</td>
<td>99%</td>
</tr>
<tr>
<td><strong>Min</strong></td>
<td>61%</td>
<td>80%</td>
<td>67%</td>
<td>91%</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>91%</td>
<td>96%</td>
<td>97%</td>
<td>99%</td>
</tr>
</tbody>
</table>
Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA ⊕ WP : combination

Improving DSE* : speed-up

<table>
<thead>
<tr>
<th></th>
<th>VA</th>
<th>WP</th>
<th>VA ⊕ WP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2.4x</td>
<td>2.2x</td>
<td>2.2x</td>
</tr>
<tr>
<td>Min</td>
<td>0.5x</td>
<td>0.1x</td>
<td>0.1x</td>
</tr>
<tr>
<td>Max</td>
<td>107.0x</td>
<td>74.1x</td>
<td>55.4x</td>
</tr>
<tr>
<td>Mean</td>
<td>7.5x</td>
<td>5.1x</td>
<td>3.8x</td>
</tr>
</tbody>
</table>

RT(1s) + LUNCOV + DSE*  

RT : random testing  
Speedup wrt DSE* alone
Conclusion

**LTest**: an all-in-one toolkit for whitebox testing of C programs

- ✓ a well-defined and expressive specification mechanism for coverage criteria
- ✓ an efficient integration into DSE
- ✓ a sound and efficient infeasibility detection