Tailored Source Code Transformations to Synthesize Computationally Diverse Program Variants

Benoit Baudry, Simon Allier, Martin Monperrus
• This talk is about the generation of very large quantities of sosie programs
sosie program

• Given a specification $S$
sosie program

• Given a specification S
• Given a program P that conforms to S
sosie program

• Given a specification S
• Given a program P that conforms to S
• A sosie of P is a variant of P that also conforms to S
Motivation

• Explore brittleness vs. plasticity of software

• Large quantities of diverse variants
  • Moving target
  • Failure detection
Software brittleness

SRSLSLRSRLLLSSRRLRL

G. Berry. « A la chasse aux bugs, la maladie du certain » (8 juin 2011)
Software brittleness hypothesis

G. Berry. « A la chasse aux bugs, la maladie du certain » (8 juin 2011)
Software brittleness hypothesis

G. Berry. « A la chasse aux bugs, la maladie du certain » (8 juin 2011)
Software brittleness hypothesis

G. Berry. « A la chasse aux bugs, la maladie du certain » (8 juin 2011)
Software brittleness
Software plasticity hypothesis
Software plasticity hypothesis

Rinard et al.
ICSE’10,
FSE’11
POPL’12,
PLDI’14

sosie

SRSLSLRSRLLSSRRLRL
SRSLSLSSRLLSSRRLRL
Specification: data and properties

fun : Function
assert abs(fun(.5) - 0.25) < 0.05
assert abs(fun(.4) - 0.16) < 0.05
assert abs(fun(.3) - 0.09) < 0.05

The test input data specifies the input domain
The assertions specify the level of abstraction
Research questions

Do sosies exist?

Can we automatically synthesize them?

What are effective transformations?
Sosiefication process

Input:
- Program P
- Specification (Test Suite)
- Program Transformation
- Variant P’
- Coverage Check
- Transformation

Output:
- metrics
- degenerated-variant P’
- Sosie P’

Steps:
1. Program Transformation
2. Transformation Configuration (optional)
3. ok
Automatic Synthesis of Sosies

- We add/deleted/replace a given statement by another one and see whether all assertions remain satisfied
  - we pick code from the same program
- Four strategies
  - random
  - wittgenstein: replace with variables that have the same name
  - reaction: replace with variables that have the same type
  - steroid: reaction + rename variables
**Experimental data**

<table>
<thead>
<tr>
<th>Package</th>
<th>#test cases</th>
<th>#assert</th>
<th>coverage</th>
<th>#statement</th>
<th>compile time</th>
<th>test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUnit</td>
<td>721</td>
<td>1535</td>
<td>82%</td>
<td>2914</td>
<td>4.5</td>
<td>14.4</td>
</tr>
<tr>
<td>EasyMock</td>
<td>617</td>
<td>924</td>
<td>91%</td>
<td>2042</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td>Dagger (core)</td>
<td>128</td>
<td>210</td>
<td>85%</td>
<td>6744</td>
<td>11.2</td>
<td></td>
</tr>
<tr>
<td>JBehave-core</td>
<td>485</td>
<td>1451</td>
<td>89%</td>
<td>4984</td>
<td>22.9</td>
<td></td>
</tr>
<tr>
<td>Metrics</td>
<td>214</td>
<td>312</td>
<td>79%</td>
<td>1471</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>commons-collections</td>
<td>1121</td>
<td>5397</td>
<td>84%</td>
<td>9893</td>
<td>22.9</td>
<td></td>
</tr>
<tr>
<td>commons-lang</td>
<td>2359</td>
<td>13681</td>
<td>94%</td>
<td>11715</td>
<td>24.6</td>
<td></td>
</tr>
<tr>
<td>commons-math</td>
<td>3544</td>
<td>9559</td>
<td>92%</td>
<td>4706</td>
<td>144.2</td>
<td></td>
</tr>
<tr>
<td>clojure</td>
<td>NA</td>
<td>NA</td>
<td>71%</td>
<td>18533</td>
<td>105.1</td>
<td>185</td>
</tr>
</tbody>
</table>
nb of trial: 298938
nb of compile: 81394
nb of sosie: 28805 (10%)
Computation diversity

• Goal: unpredictability of execution flow

• Computation monitoring:
  • method calls diversity
  • variable diversity
Easymock: 465 sosies
Dagger: 481 sosies
Junit: 446 sosies
Conclusion

• Sosies exist
  • for all programs

• Sosies can exhibit computation diversity

• Next steps
  • variability-aware execution
  • is computational diversity unbounded?

https://github.com/DIVERSIFY-project/sosies-generator
http://diversify-project.eu/sosiefied-programs/
References

• Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A. Kelner, Martin C. Rinard: Randomized accuracy-aware program transformations for efficient approximate computations. POPL 2012: 441-454

• Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, Westley Weimer: Post-compiler software optimization for reducing energy. ASPLOS 2014: 639-652

Sosies on line

- MDMS
  - simple blog app
  - JS on client and server sides
- Server side stack
  - JS
  - Java
  - DB
  - environment
Sosies on line

- Monoculture
  - multiple instances for performance
  - load balancer
  - all instances are clones
Sosies on line

- Diversified deployment
  - All server instances are different
  - Combine natural and artificial diversity
Reactions graph

- One node per reaction
- There is an edge between n1 and n2 if
  \[ n2.in\_context == n1.in\_context \lor n1.out\_context \]
Two reactions graph (apache.common)

• Statement reactions graph
  • #edges = 12304
  • #nodes = 863
  • graph-diameter = 3
  • avg path length = 1.466
  • avg degree = 14.257

• Expression reactions graph
  • #edges = 37650
  • #nodes = 1953
  • graph-diameter = 4
  • avg path length = 1.162
  • avg degree = 19.278