Heuristics-based Parameter Generation for Java Methods

Michael Kuperberg (mkuper@ipd.uka.de)
Fouad Omri (fouad.omri@stud.uni-karlsruhe.de)

Chair “Software Design and Quality”, Faculty of Informatics,
Universität Karlsruhe (TH), Germany
Motivation

- Example: generate parameters for Java API method `String.substring(int index)` for benchmarking
  - Parameter constraints: `index ≥ 0` and `index < str.length()`
  - Need a non-null invocation target (`String` instance, e.g. `str`)

- Constraints clear to humans – unavailable to machines
- Violation of constraints → exceptions at runtime

- Java platform API: thousands of methods
  - Manual parameter generation: too costly
  - Random parameter generation: ignores above constraints
Related Work
Automated Parameter Generation

- Eclat [7]: random generation, based on execution results
  - needs an existing correct execution to start generation

- RANDOOP [8] extends Eclat, e.g. with a pool of values
  - from which an initial random input is computed

- Godefroid et al. [9]: a symbolic execution approach
  - uses random input generation (also called ”concolic execution”)

- Majumdar and Xu [10] say: [7-9] are problematic in practice
  - randomly generated parameters are in most cases meaningless for the program execution (out of scope; lead to exceptions)
  - hence, valid inputs must be specified by a context-free grammar
Challenges

- Input parameter values have to be generated in accordance to the declared static types

- "Easy": primitive static types (int, long, double, etc.)
- "Hard": reference static types, e.g. interface types
  - E.g. cs in String.contentEquals(CharSequence cs)
  - No methods to get implementors of an interface in Java

- If method is part of an API: reuse API structure?
  - E.g. scan the API to find CharSequence implementers
  - Java has no corresponding concepts in the Reflection API
HeuriGenJ: Overview

- Heuristics-based parameter Generation for Java
Parameter Type Graph

- Example:
  - Parameter type graph built by indexing implementations
    - we have tools to index bytecode and source code
    - incl. method meta-data (obtained using Java Reflection API)
  - Knowledge is encoded as Prolog-like inference rules:
    - String <- String(char[]), String(byte[]),...
HeuriGenJ: Overview

- Heuristics-based parameter Generation for Java

---

**API Structure Analyser**

1. Parse API implementation
2. Build an API model and a parameter graph

**Learning System**

- Data repository
- 5. Analyse and store execution information

**Heuristical Parameter Generator**

- 3. Generate/Retrieval A Set of Input Parameters
- 4. Execute method

**Heuristical Exception Handler**

- 6. Estimate Causes
- 7. Modify Input for Heur. Parameter Generator

---

Steps 3-5 repeated for each considered method

Proceed with next method from step 3

Successful?

YES

NO (an exception occurred)
Handling Parameter-caused Exceptions

- Exception causes determination and evaluation:
  - example: an `ArrayIndexOutOfBoundsException` ⇒ an `int`-typed parameter shall match an array parameter
  - **HeuriGenJ formally encodes such rules**
  - heuristically estimating *which* `int`-typed parameter in a signature is causing the exception

- In general, invocation targets are also considered
  - e.g. `IndexOutOfBoundsException` for `str.substring(int index)` → create `str` with sufficient length
Heuristical Parameter Generator (HPG)

- Parameter generation is nondeterministic
- A set of heuristics was created to avoid exceptions
- Basic idea: formalise rules of parameter constraints

- A multi-stage heuristic for generation of int primitives:
  - Use the constants defined in the declaring class
  - Choose a value from an interval: lower bound = 0
  - Upper bound := min length of all container types in signature
  - Upper bound := length of the invocation target
  - A random positive int value
  - Upper bound := constant int values declared by the class or its subclasses
Evaluation

- Visible, implemented methods: public, non-abstract

- **Coverage**: the number and share of methods for which arguments are successfully generated
  - without human intervention or external data sources

- **Effectiveness**: the number and share of exceptions at runtime successfully handled by HeuriGenJ

- **Effort**: the duration of the parameter generation

- **Targets**: `java.util` and `java.lang` API packages

- **Setting**: Intel Pentium Dual-Core 1.8 GHz, 1 GB main memory, Windows Vista with JRE 1.6.0_03
## Evaluation

<table>
<thead>
<tr>
<th>Package</th>
<th>#Public non-abstract classes</th>
<th>#Public non-abstract methods</th>
<th>#Executed methods w/o exceptions</th>
<th>Execution success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util</td>
<td>58</td>
<td>738</td>
<td>668</td>
<td>90.51%</td>
</tr>
<tr>
<td>java.lang</td>
<td>76</td>
<td>861</td>
<td>790</td>
<td>91.75%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Package</th>
<th>Total duration of arguments generation</th>
<th>#Thrown runtime exceptions</th>
<th>#Handled runtime exceptions</th>
<th>Exception handling success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util</td>
<td>169 s</td>
<td>160</td>
<td>95</td>
<td>59.37%</td>
</tr>
<tr>
<td>java.lang</td>
<td>259 s</td>
<td>204</td>
<td>151</td>
<td>74.01%</td>
</tr>
</tbody>
</table>
Related Work
Automated Parameter Generation

- Eclat [7]: random generation, based on execution results
  - needs an existing correct execution to start generation

- RANDOOP [8] extends Eclat, e.g. with a pool of values
  - from which an initial random input is computed

- Godefroid et al. [9] present a symbolic execution approach
  - uses random input generation (also called ”concolic execution”)

- Majumdar and Xu [10]: [7-9] are problematic in practice
  - randomly generated parameters are in most cases meaningless for the program execution (out of scope; lead to exceptions)
  - hence, valid inputs must be specified by a context-free grammar
Assumptions & Limitations

Assumptions
- Exceptions must be available to HeuriGenJ
- SQL-like String parameters cannot be generated

Limitations
- Parameter generation not applied to methods/APIs
  - ... if they may destroy sensitive data (e.g. using JDBC)
  - ... if they produce a security issue or undesired effects (e.g. the java.lang.System.exit method)
  - Excluding such API parts is not automated in HeuriGenJ
- Usability in benchmarking: to be demonstrated
Future Work

- Generate parameters w.r.t. quantification of parameter performance dependencies
  - several input sets per signature (sensitivity analysis)
  - vary invocation target objects

- Enhance the heuristical parameter generation
  - by incorporating machine learning techniques
  - using search-based software engineering techniques
  - use recorded parameters (e.g. with ByCounter [11])

- Integrate into benchmarking/perf. prediction
- Port to .NET CLR and other environments
Conclusions

- **HeuriGenJ**: a novel approach for input parameter generation using heuristics
  - For methods and constructors with implementation in Java bytecode
  - Handles runtime exceptions if they occur and learns from them
  - Saves execution information and parameter values in a database
  - The principles of HeuriGenJ applicable to other languages and APIs, e.g. .NET platform API

Thanks for your attention! Questions?
References


