Balancing Soundness and Efficiency for Practical Testing of Configurable Systems

Sabrina Souto  
UEPB, Brazil  
sabrinadfs@gmail.com

Marcelo d’Amorim  
UFPE, Brazil  
damorim@cin.ufpe.br

Rohit Gheyi  
UFCG, Brazil  
rohit@dsc.ufcg.edu.br
Configurable Systems

Configurable System

class Code{
    if (OPTION_1){
        if (OPTION_2){
            //...
        }
    }
    if (OPTION_3){
        //...
    }
}

Configurations

Many other examples!
Bugs in Configurable Systems

Configurable System

System Configurations

- Configuration-related bug!

...
Testing Configurable Systems

Monolithic System

Tests

System Configurations

Tests

Combinatorial explosion!
Limitations of Existing Techniques

![Graph showing Efficacy (#failures) vs. Efficiency (#samples)]
Limitations of Existing Techniques

Exhaustive

* Find all bugs

Very expensive
Limitations of Existing Techniques

- **Exhaustive**
  - Find all bugs
  - Very expensive

- **Default**
  - Very efficient
  - Can miss bugs

- **Efficacy (#failures)**
- **Efficiency (#samples)**
Limitations of Existing Techniques

**Exhaustive**
- Find all bugs
- Very expensive

**Sampling**
- Try to find bugs with less samples
- False positives and false negatives

**Default**
- Very efficient
- Can miss bugs
Limitations of Existing Techniques

Efficacy (#failures) vs Efficiency (#samples)

- **Exhaustive**
  - Find all bugs
  - Very expensive

- **Dynamic (SPLat [FSE’13,SPLC’15])**
  - Consider code and test
  - It may not scale in all cases

- **Sampling**
  - Try to find bugs with less samples
  - False positives and false negatives

- **Default**
  - Very efficient
  - Can miss bugs
Limitations of Existing Techniques

Efficacy (#failures)

* Exhaustive
* Dynamic (SPLat)

S-SPLat
Sampling + SPLat

* Sampling

* Default

Efficiency (#samples)
Example

<table>
<thead>
<tr>
<th>SPLat</th>
<th>S-SPLat (one-enabled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling (one-enabled)</td>
<td></td>
</tr>
</tbody>
</table>
Example

Notepad
- **17** configuration variables
- Only 3 are reached by toolbar()

```java
class Notepad {
    void toolbar() {
        if (TOOLBAR) {
            //...
            if (WORDCOUNT) {
                //...
            }
        }
        if (MENU BAR) {
            //...
        }
        //...
    }
}
```
Example

Notepad
- 17 configuration variables
- Only 3 are reached by toolbar()

```
class Notepad {
    void toolbar() {
        if (TOOLBAR) {
            //...
            if (WORDCOUNT) {
                //...
            }
        }
        if (MENUBAR) {
            //...
        }
        //...
    }
}
```
Example

Notepad
- **17** configuration variables
- Only 3 are reached by toolbar()

```java
class Notepad {
    void toolbar() {
        if (TOOLBAR) {
            //...
            if (WORDCOUNT) {
                //...
            }
        }
        if (MENUBAR) {
            //...
        }
        //...
    }
}
```

Sampling (one-enabled)
- 17 configurations

SPLat
- 6 configurations

S-SPLat (one-enabled)
Example

Notepad
- 17 configuration variables
- Only 3 are reached by toolbar()

```java
class Notepad {
    void toolbar() {
        if (TOOLBAR) {
            //...
            if (WORDCOUNT) {
                //...
            }
        }
        if (MENUBAR) {
            //...
        }
        //...
    }
}
```
S-SPLat

**Input**
- Instrumented Configurable System
- Tests
- Sampling Heuristic
- Feature Model (Optional)

**Output**
Tests executed with **reachable** and **satisfiable** configurations

- C1, T1
- C2, T1
- C1, T2
- C5, T2
- C4, T3
- ...
- ...

16
S-SPLat

Input
- Instrumented Configurable System
- Tests
- Sampling Heuristic
- Feature Model (Optional)

For all tests

- Run the test $T_i$

  Find reachable variables

  Look for next reachable configuration

  Check:
  - Sampling heuristic
  - Feature model

Output
- Tests executed with reachable and satisfiable configurations:
  - $C_1, T_1$
  - $C_2, T_1$
  - $C_1, T_2$
  - $C_5, T_2$
  - $C_4, T_3$
  - ...
  - ...

Yes

Otherwise
EVALUATION
Research Questions

RQ1 → Which heuristics maximize efficiency (#samples)?

RQ2 → Which heuristics maximize efficacy (#failures)?

RQ3 → Which heuristics (basic or combination) maximize efficiency and efficacy?
Scenarios

Software Product Lines (SPLs)

8 subjects
- All existing tests
- All existing options

Version 6.1
- 3,557 tests
- 50 most frequently cited options in bug reports

Version 4.8.2

 GCC

- 17K+ tests
- 2k+ variables
Evaluation

SPLs
Evaluation Techniques

SPLs

8 subjects

Techniques:
1. SPLat
2. SPLat + med
3. SPLat + oe
4. SPLat + od
5. SPLat + pw
6. SPLat + ran

[ICSE’16, ASE’14]
Findings

**RQ1:** Which heuristics maximize efficiency (#samples)?

- SPLat and SPLat+

- SPLat+

**RQ2:** Which heuristics maximize efficacy (#failures)?

- SPLat+

- SPLat+

- SPLat+
Findings

RQ3: Which heuristics maximize efficiency (#samples) and efficacy (#failures)?

Combinations of heuristics

- oe x od x med x pw
- c1 = oe+od
- c2 = oe+med
- c3 = oe+pw ...
- c11 = oe+od+med+pw
Findings

RQ3: Which heuristics maximize efficiency (#samples) and efficacy (#failures)?

- SPLat+Most-enabled-disabled (oe + od + med + pw) optimized #failures at the expense of #samples
- SPLat+c11 optimized #samples at the expense of #failures
- SPLat did not scale for some subjects

The sampling heuristics reduced the number of samples explored by SPLat yet retaining their ability to reveal failures.
Evaluation
Evaluation Techniques

Techniques:
1. SPLat
2. SPLat + med
3. SPLat + oe
4. SPLat + od
5. SPLat + pw
6. SPLat + ran

[ICSE’16, ASE’14]
Findings

**RQ1:** Which heuristics maximize efficiency (#samples)?

- **SPLat+**
- **Random (ran)**

- **SPLat+**
- **Most-enabled-disabled (med)**

- **SPLat+** and **SPLat+**
- **Pair-wise (pw)**

**RQ2:** Which heuristics maximize efficacy (#failures)?

- **SPLat+**
- **One-enabled (oe)**

- **SPLat+**
- **One-disabled (od)**
Findings

**RQ3:** Which heuristics maximize efficiency (#samples) and efficacy (#failures)?

It is preferable to pick the best performing heuristics in the leftmost group → the best choices!
Findings

**RQ3:** Which heuristics maximize efficiency (#samples) and efficacy (#failures)?

All five bugs were captured. **SPLat+c2** (oe+med) found all bugs with a relatively small number of samples.
Lessons Learned

- For SPLs → c11 (oe+od+med+pw)
- For GCC → c2(oe+med)
- For SPLs and GCC → c7 (oe+od+med) [ICSE 2016] A comparison of 10 sampling algorithms for configurable systems.
- Combine different simple heuristics
- Avoid heuristics with a large number of requirements
S-SPLat found a good balance between bugs and samples
The sampling heuristics helped to reduced the number of samples explored by SPLat without loss the ability to find bugs
S-SPLat could deal with scalability
It revealed bugs in potentially large configuration spaces

https://sabrinadfs.github.io/s-splat/
sabrinadfs@gmail.com
BACKUP SLIDES
**RQ1: #samples**

- **SPLat** and **ran** explored much samples.
- **med** explored the smallest sample sets.
- **od** explored the largest sample sets.

**RQ2: #failures**

- **od** and **pw** found almost the same number of failures as **splat** but they required much fewer samples.
RQ3: \#samples x \#failures

- Combinations of heuristics
  - \texttt{oe} x \texttt{od} x \texttt{med} x \texttt{pw}
    - \texttt{c1} = oe+od
    - \texttt{c2} = oe+med
    - \texttt{c3} = oe+pw...
    - \texttt{c11} = oe+od+med+pw

\textbf{SPLat} and \texttt{med} optimize one dimension at the expense of the other.

\textbf{c11} (oe + od + med + pw) performed consistently well in all cases.

The sampling heuristics reduced the number of samples explored by SPLat yet retaining their ability to reveal failures.
RQ1: #samples

pw found more failures. It was one of the most expensive techniques.

RQ2: #bugs

oe and od found almost the same number of failures as pw but with much fewer samples.
Discussion

• **c2** found all crashes with a relatively low number of configurations

• **c7** performed better, it detected most failures and crashes through a relatively small number of configurations

• Combine different simple heuristics instead of using one that entails a larger number of test requirements

• **S-SPLat** is promising to reveal errors in potentially large configuration spaces
Handling Constraints

**SPLs**

Complex models
- 54% of the selected configurations are invalid
- 43% of failures are false positives

**GCC**

The use of validation is not necessary
- Crashes was only revealed in valid configurations

The techniques performed consistently with and without feature constraints
Additional Evaluations

S-SPLat x Regular Sampling

Regular Sampling detected the same bugs as S-SPLat with more configurations.

Random Sampling with more rates: 10% and 30%

New results are proportional to the change in the sampling rates of random.
Threats to Validity and Limitations

• The selection of subjects
  • We used subjects from a variety of sources, including a large configurable system with hundreds of options

• Eventual implementation errors
  • We thoroughly checked our implementation and our experimental results
  • Our datasets and implementations are publicly available: https://sabrinadfs.github.io/s-splat/

• SPLat currently only supports systems with dynamically bound feature variables ])
  • It remains to investigate how SPLat and S-SPLat would perform on systems with #ifdef variability