Balancing Soundness and Efficiency for Practical Testing of Configurable Systems

Sabrina Souto
UEPB, Brazil
UEPB
sabrinadfs@gmail.com

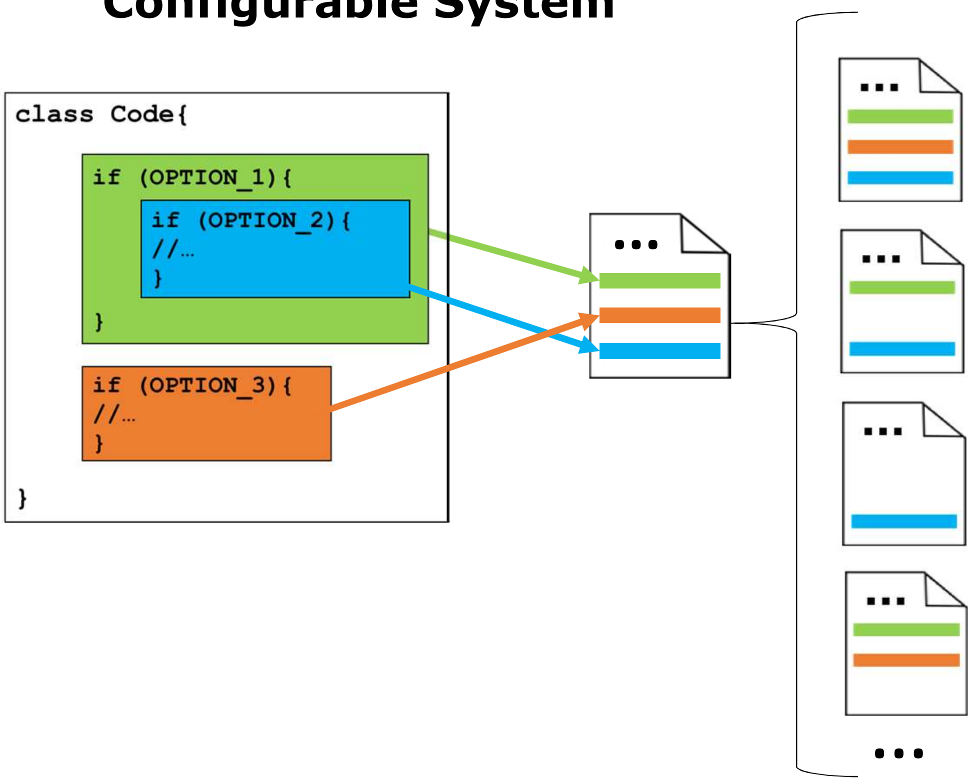
Marcelo d'Amorim
UFPE, Brazil
UFPE
damorim@cin.ufpe.br

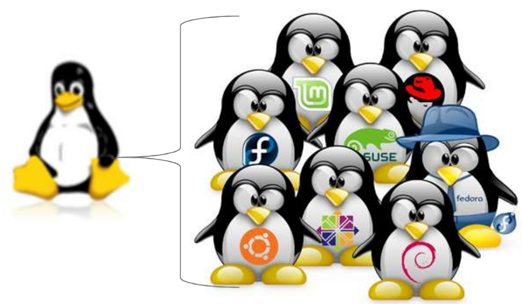
Rohit Gheyi
UFCG, Brazil
UFCG
rohit@dsc.ufcg.edu.br

Configurable Systems

Configurations

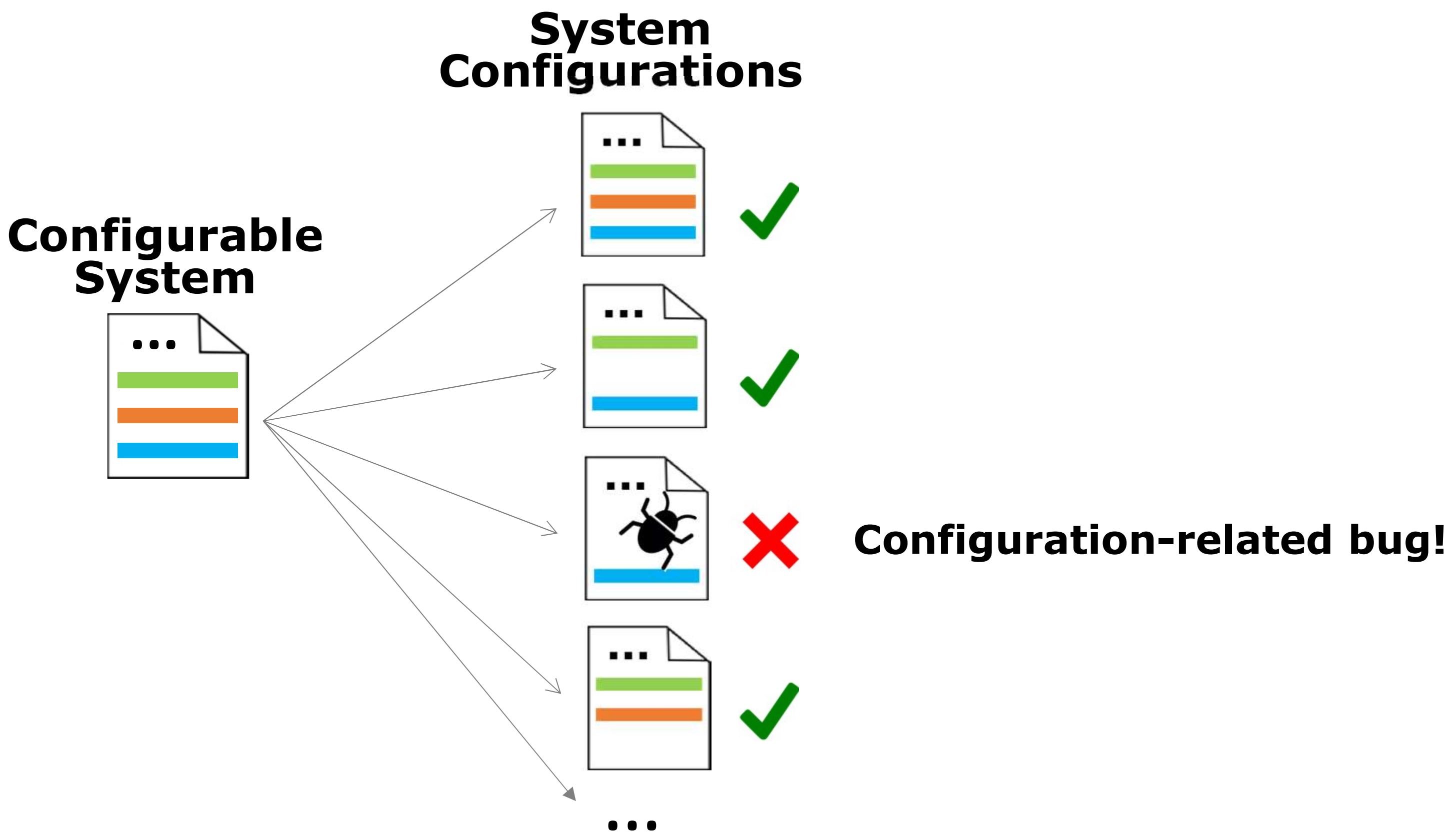
Configurable System



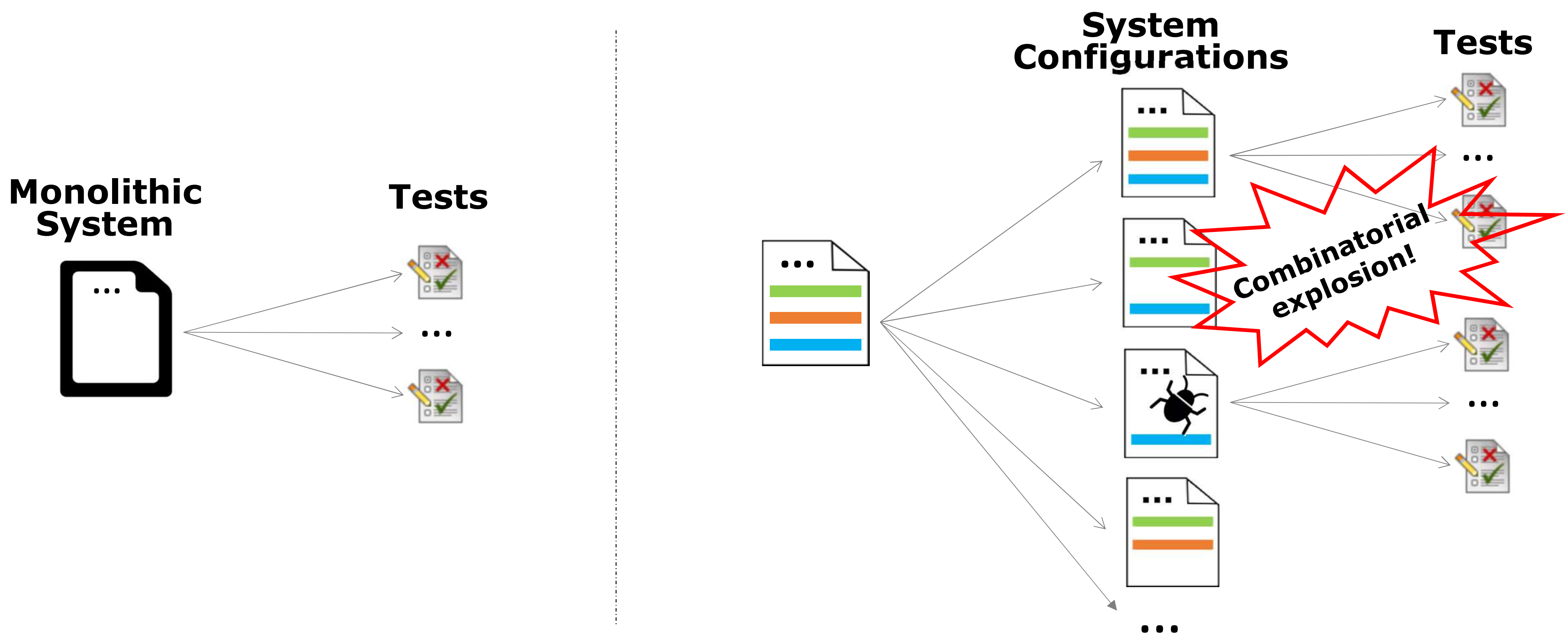


Many other examples!

Bugs in Configurable Systems

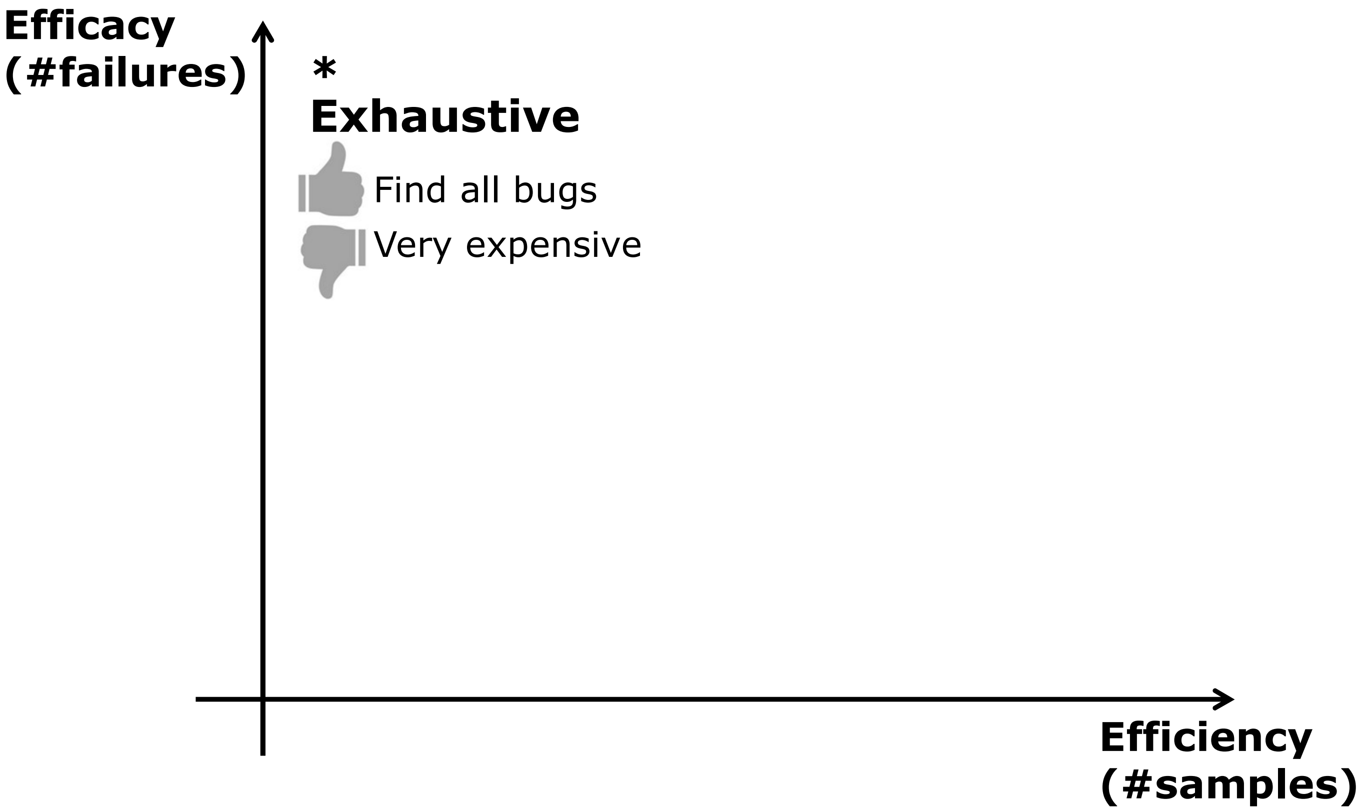


Testing Configurable Systems

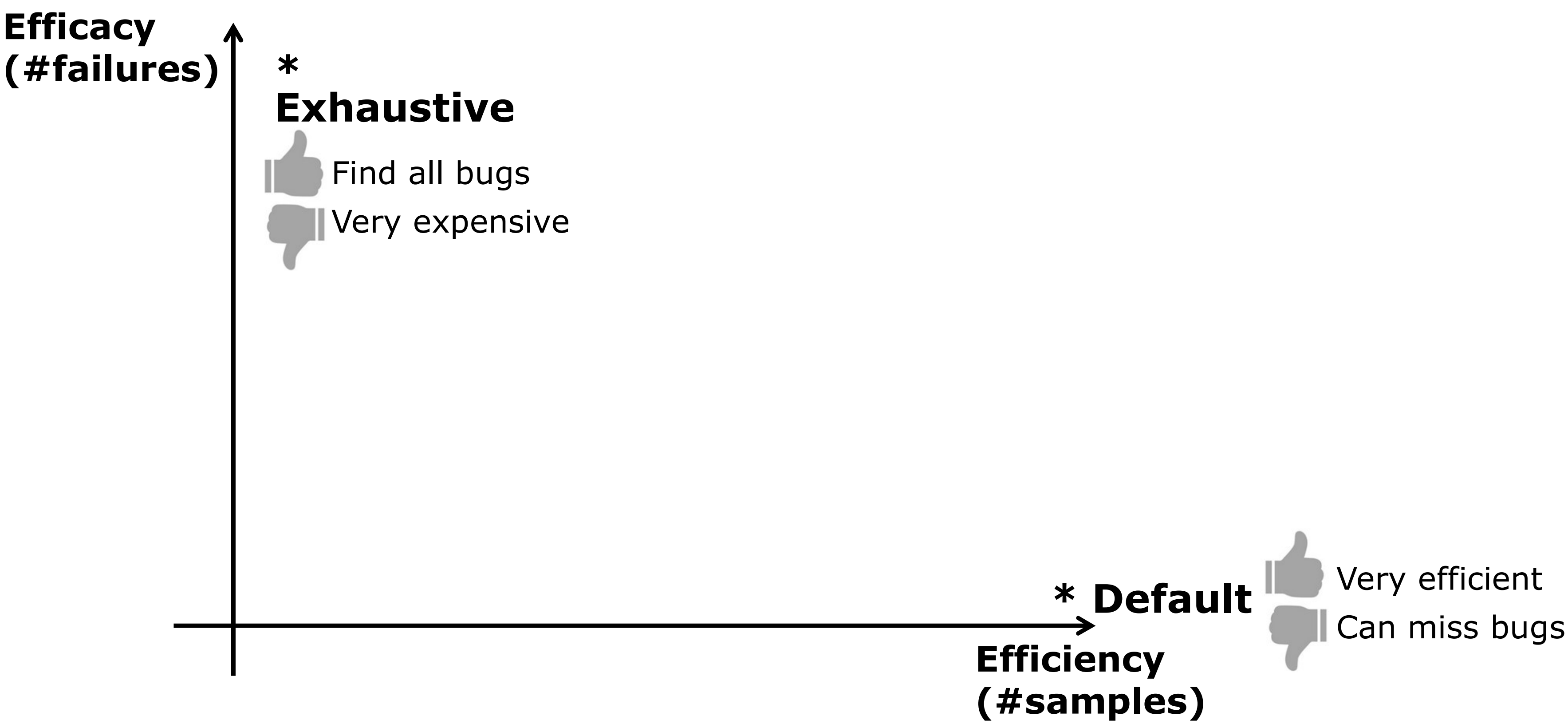


Limitations of Existing Techniques

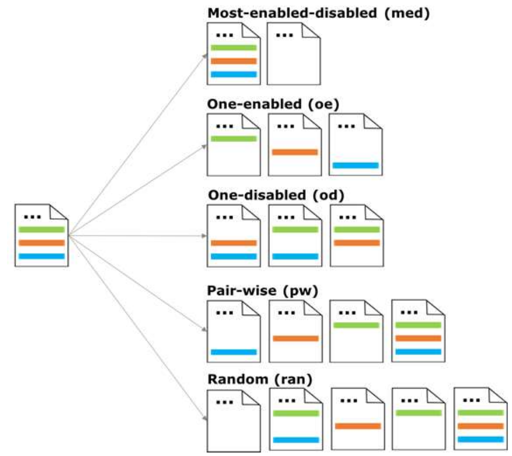
Limitations of Existing Techniques



Limitations of Existing Techniques



Limitations of Existing Techniques



Efficacy (#failures)

*** Exhaustive**

Find all bugs

Very expensive

*** Sampling**

Try to find bugs with less samples

False positives and false negatives

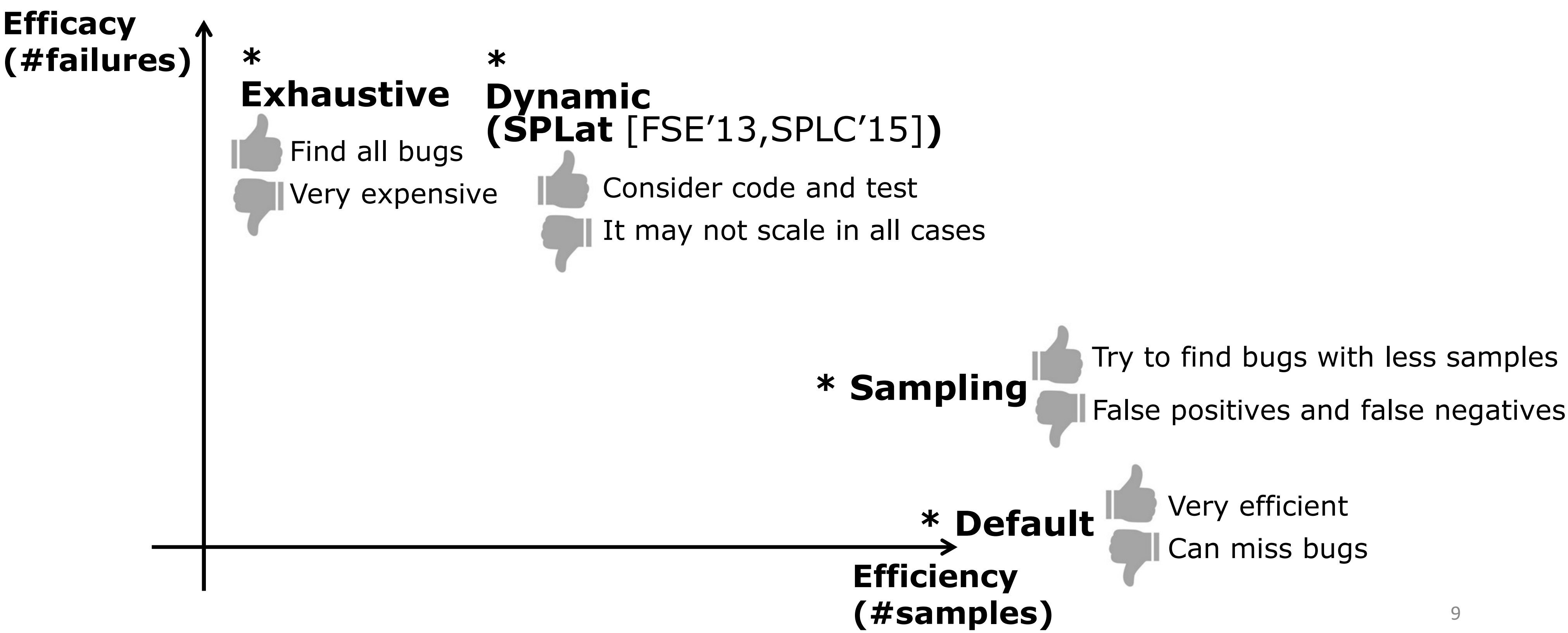
*** Default**

Very efficient

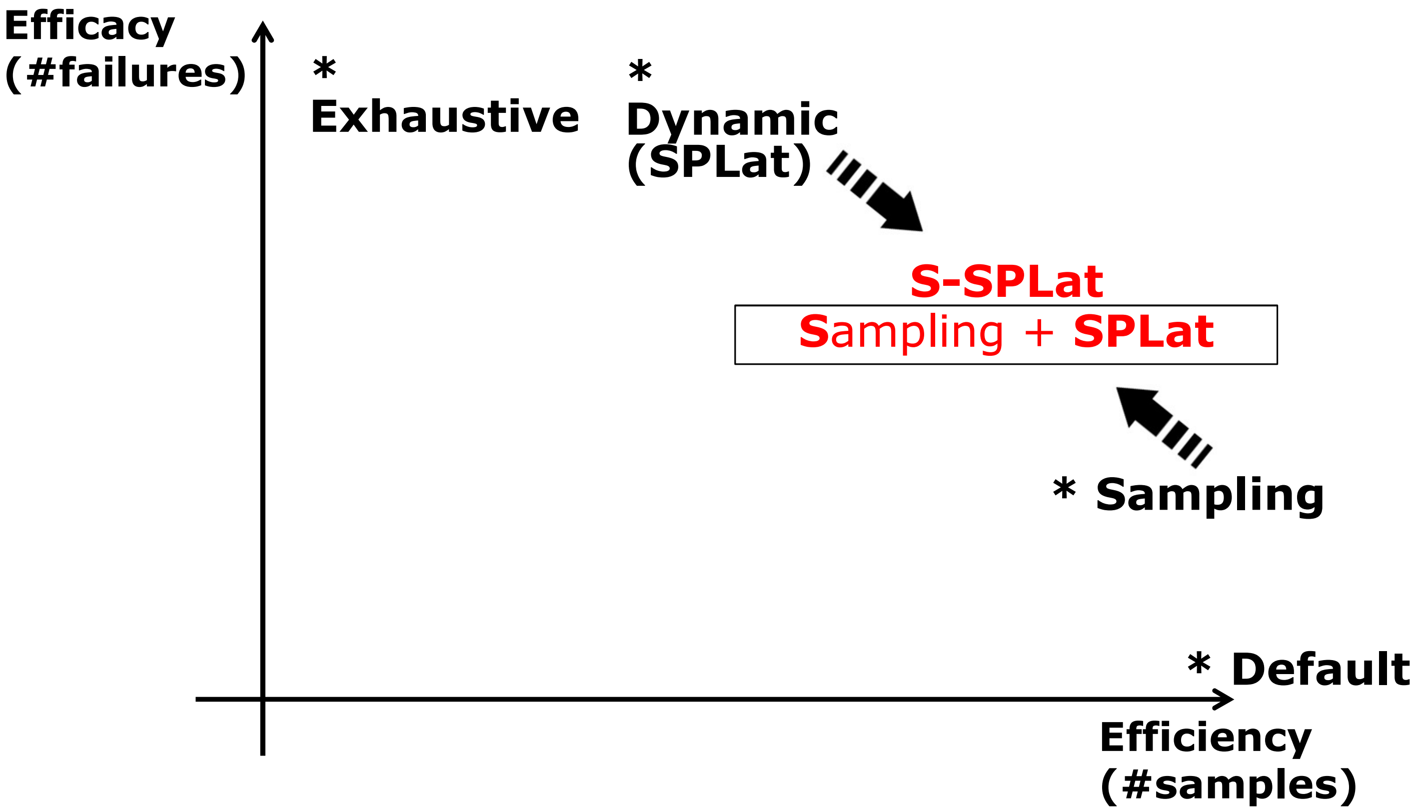
Can miss bugs

Efficiency (#samples)

Limitations of Existing Techniques



Limitations of Existing Techniques



Example

Sampling (one-enabled)

SPLat

S-SPLat (one-enabled)

Example

Notepad

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

Test

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

        if (MENUBAR) {
        //...
        }
    }
    //...
}
```

Sampling (one-enabled)

SPLat

S-SPLat (one-enabled)

Example

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

Test

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

        if (MENUBAR) {
        //...
        }
    }
    //...
}
```

Sampling (one-enabled)

17 configurations

SPLat

S-SPLat (one-enabled)

Example

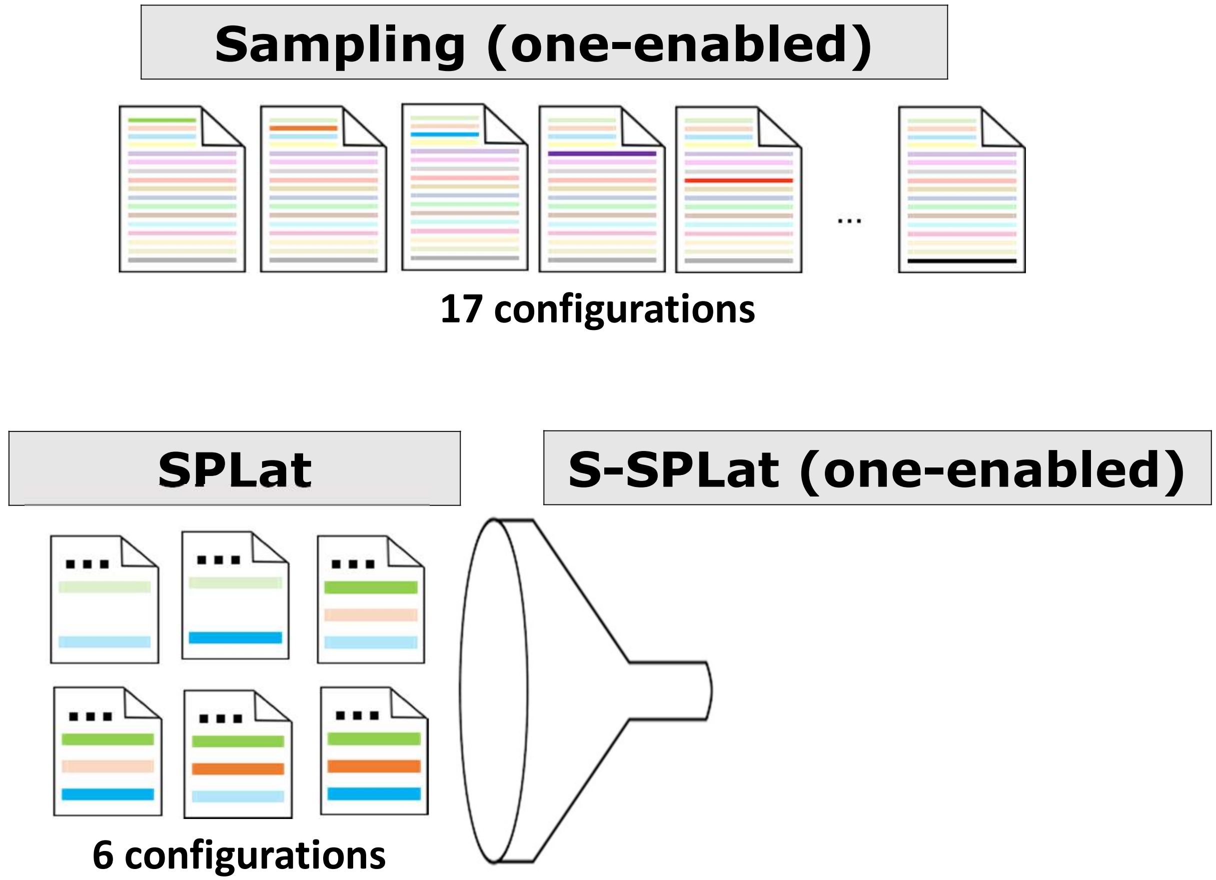
Notepad

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

Test

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

    if (MENUBAR) {
    //...
    }
  }
  //...
}
```



Example

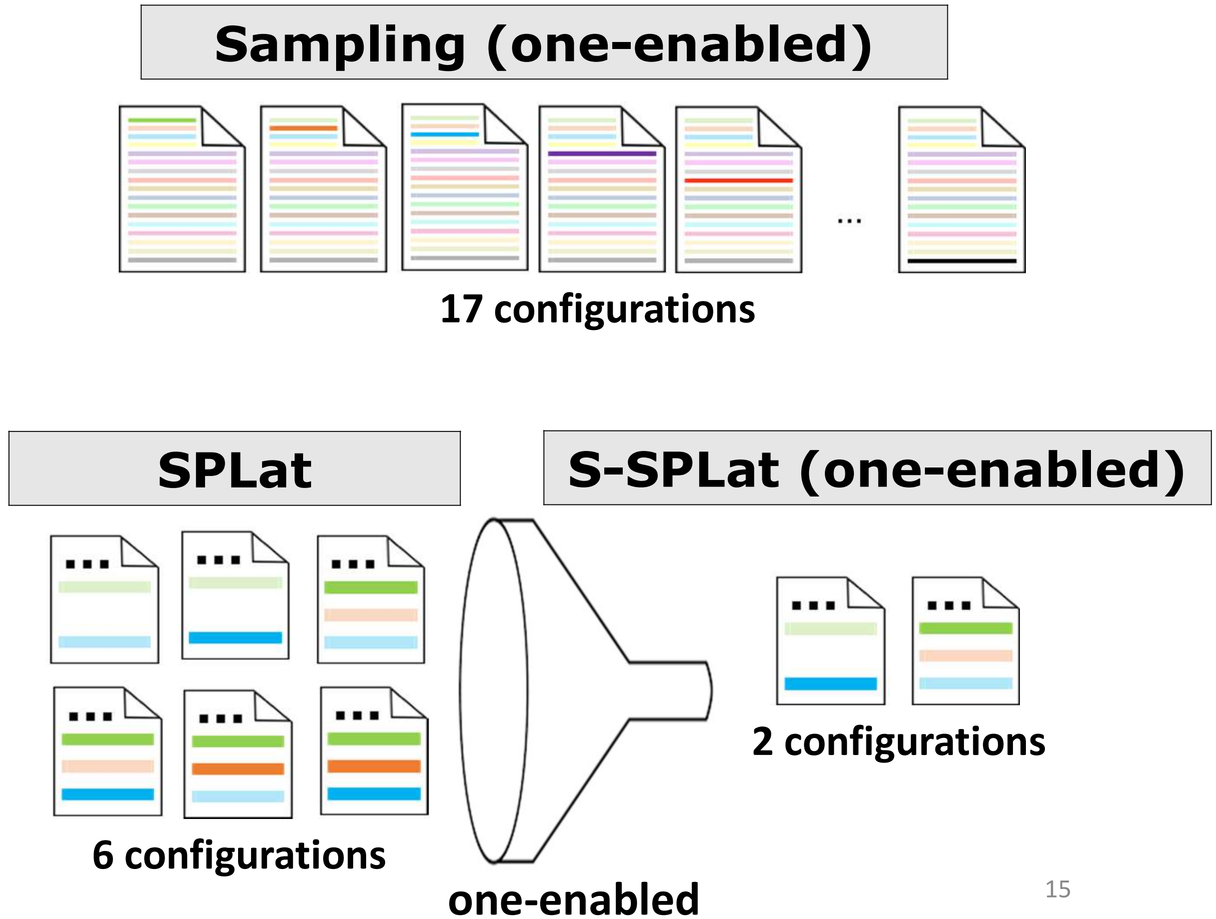
Notepad

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

Test

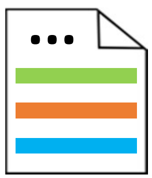
```
class Notepad {
  void toolbar() {
    if (TOOLBAR) {
      //...
      if (WORDCOUNT) {
        //...
      }
    }

    if (MENUBAR) {
    //...
    }
  }
  //...
}
```



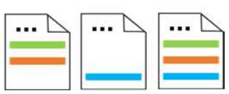
S-SPLat

Input

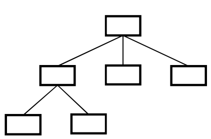


Instrumented Configurable System

Tests



Sampling Heuristic

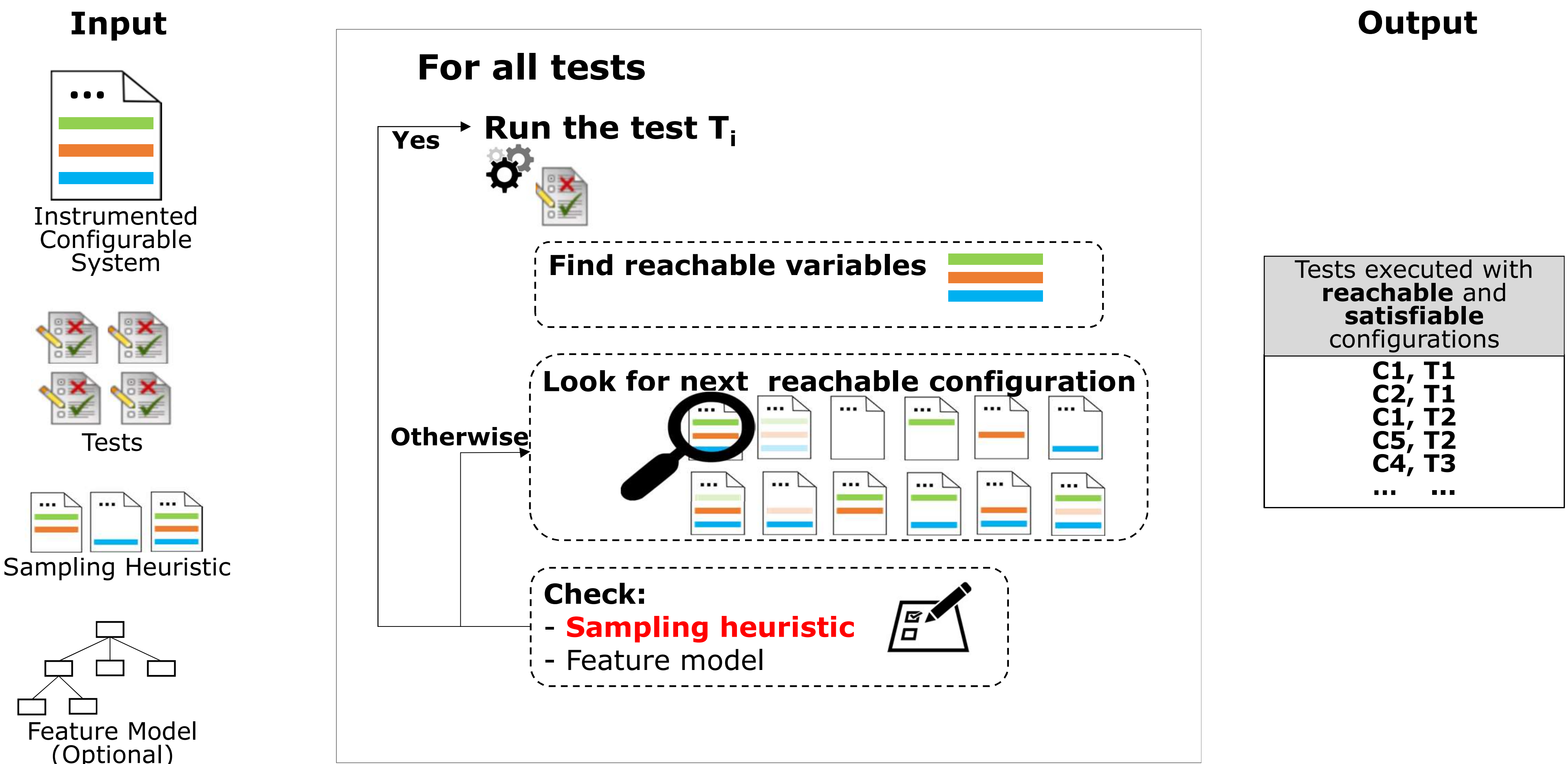


Feature Model (Optional)

Output

Tests executed with **reachable** and **satisfiable** configurations
C1, T1
C2, T1
C1, T2
C5, T2
C4, T3
… …

S-SPLat



Input
Instrumented Configurable System
Tests
Sampling Heuristic
Feature Model (Optional)
For all tests
Yes
Run the test T_i
Find reachable variables
Otherwise
Look for next reachable configuration
Check:
- Sampling heuristic
- Feature model
Output
Tests executed with **reachable** and **satisfiable** configurations
C1, T1
C2, T1
C1, T2
C5, T2
C4, T3
... ...

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

- **17K+ tests**
- **2k+ variables**

GCC

Software Product Lines (SPLs)

8 subjects

- All existing tests
- All existing options

Version 6.1

Version 4.8.2

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

Evaluation

SPLs

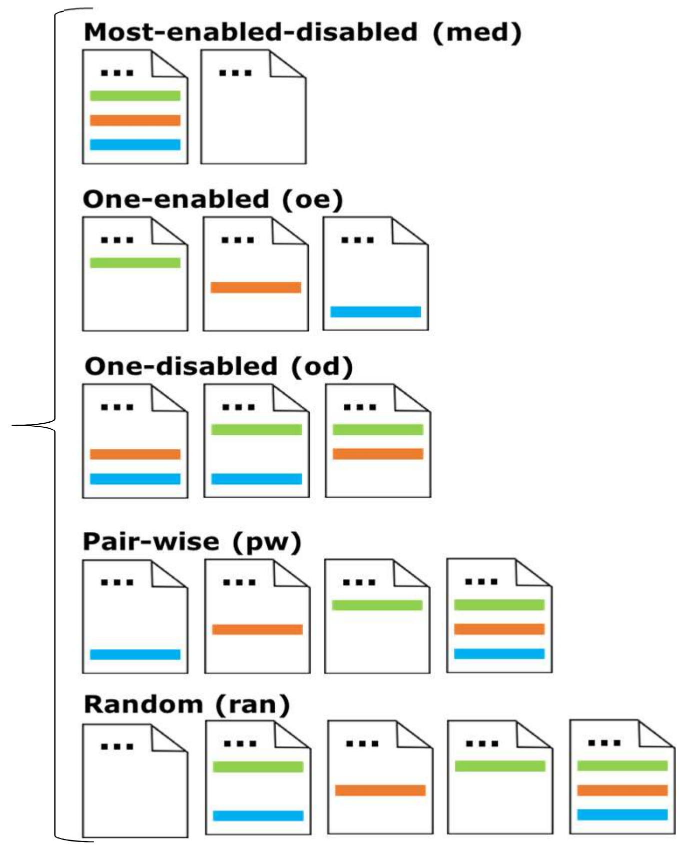
Evaluation Techniques

[ICSE’16,ASE’14]

SPLs

8 subjects

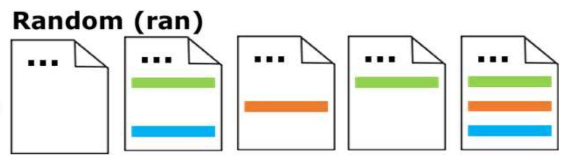
Techniques:
1. SPLat
2. SPLat + med
3. SPLat + oe
4. SPLat + od
5. SPLat + pw
6. SPLat + ran



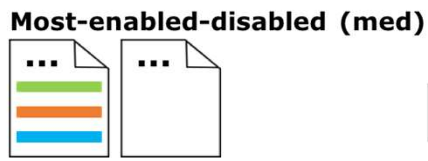
Findings

RQ1: Which heuristics maximize efficiency (#samples)?

SPLat and **SPLat+**



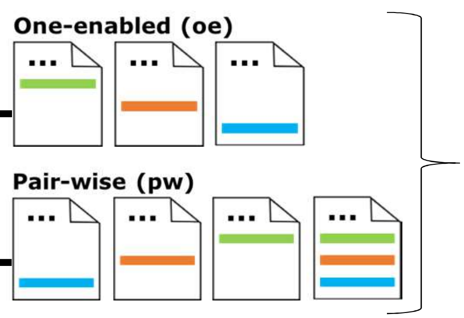
SPLat+



RQ2: Which heuristics maximize efficacy (#failures)?

SPLat+

SPLat+



Findings

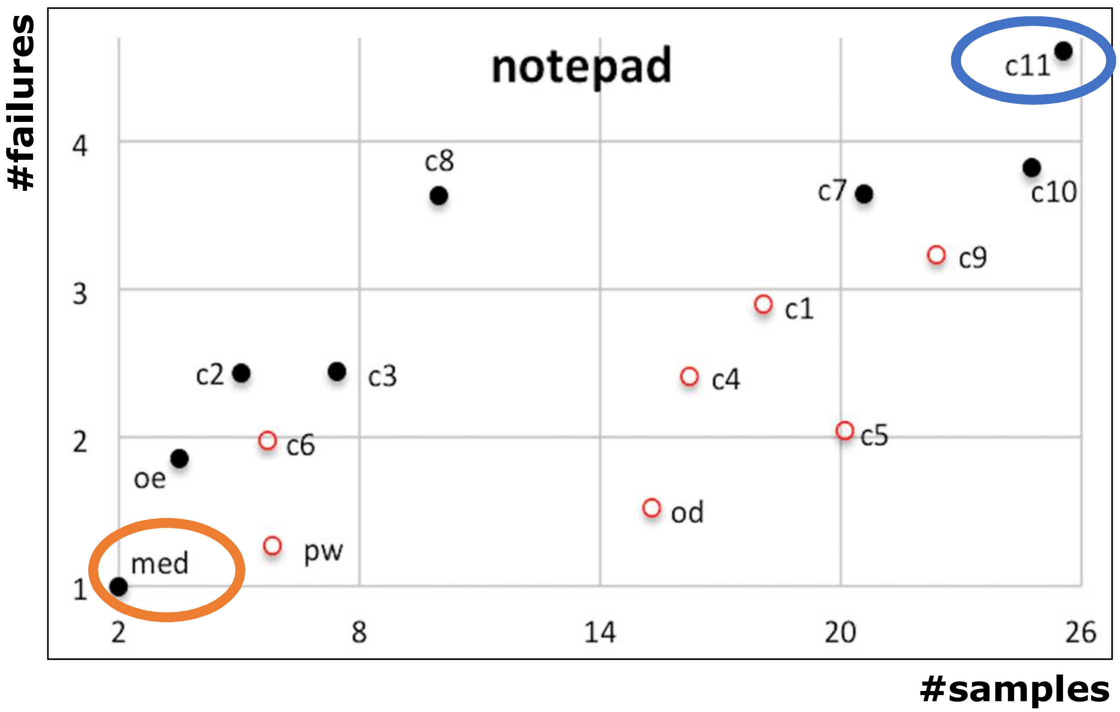
RQ3: Which heuristics maximize efficiency (#samples) and efficacy (#failures)?

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** optimized #samples at the expense of #failures
- **SPLat+c11** (oe + od + med + pw) optimized #failures at the expense of #samples
- **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

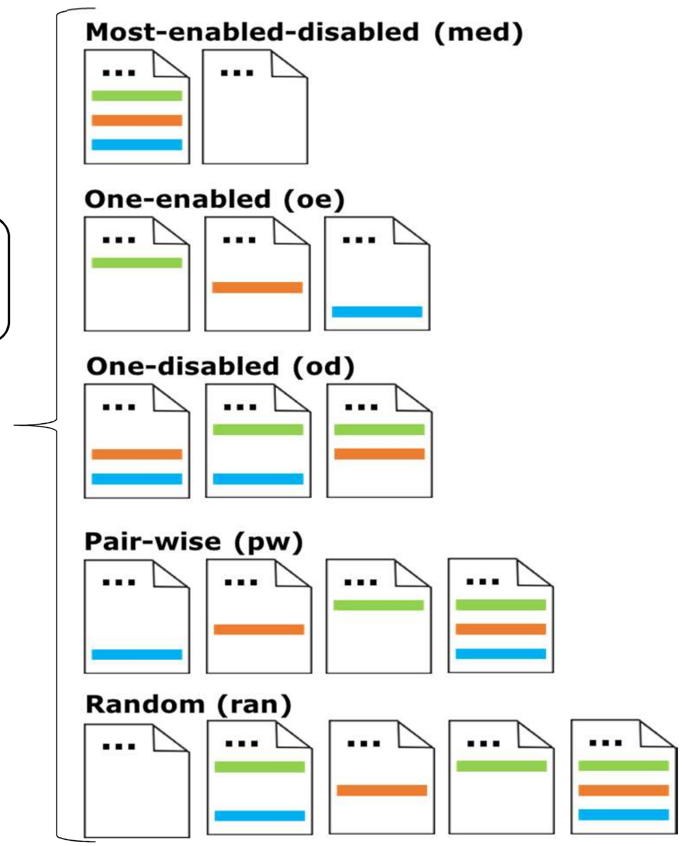
[ICSE'16,ASE'14]

Version 6.1

Version 4.8.2

Techniques:

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



Findings

Evaluation SPLs

Evaluation

GCC

Version 6.1

RQ1: Which heuristics maximize efficiency (#samples)?

Random (ran)

SPLat+ and **SPLat+**

Pair-wise (pw)

Most-enabled-disabled (med)

SPLat+

RQ2: Which heuristics maximize efficacy (#failures)?

One-enabled (oe)

SPLat+

One-disabled (od)

SPLat+

Findings

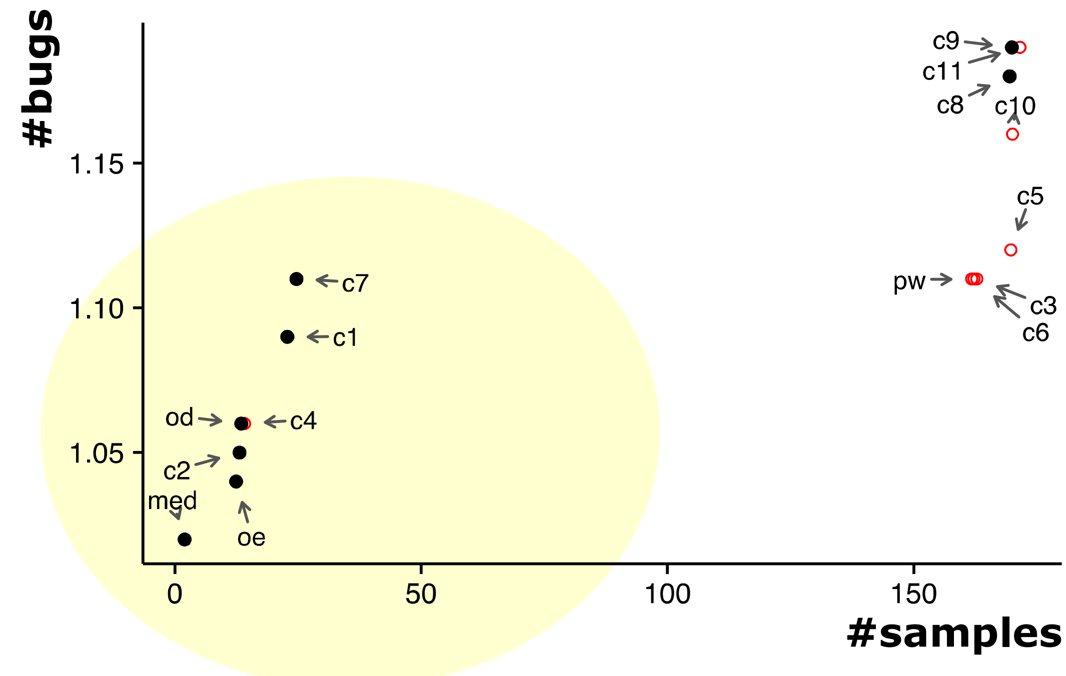
Evaluation SPLs

Evaluation

GCC

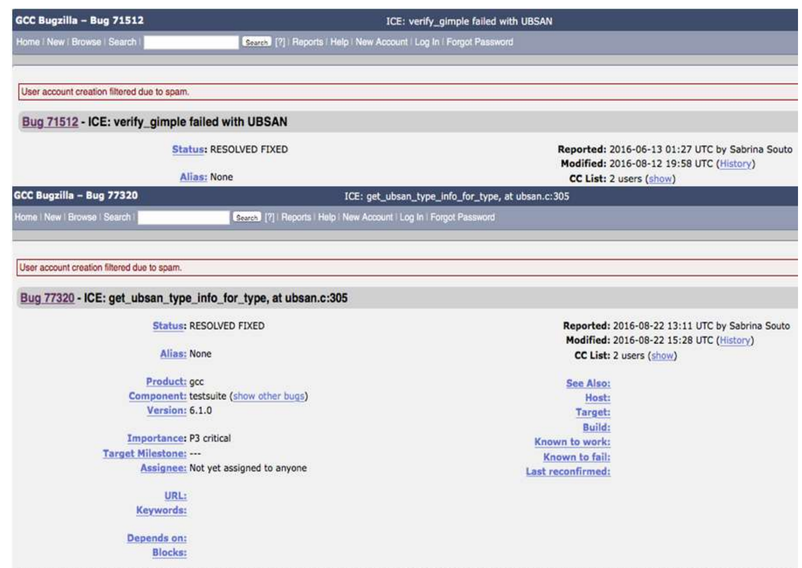
Version 6.1

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**!

Bugs found



2 new bugs reported.

Evaluation SPLs

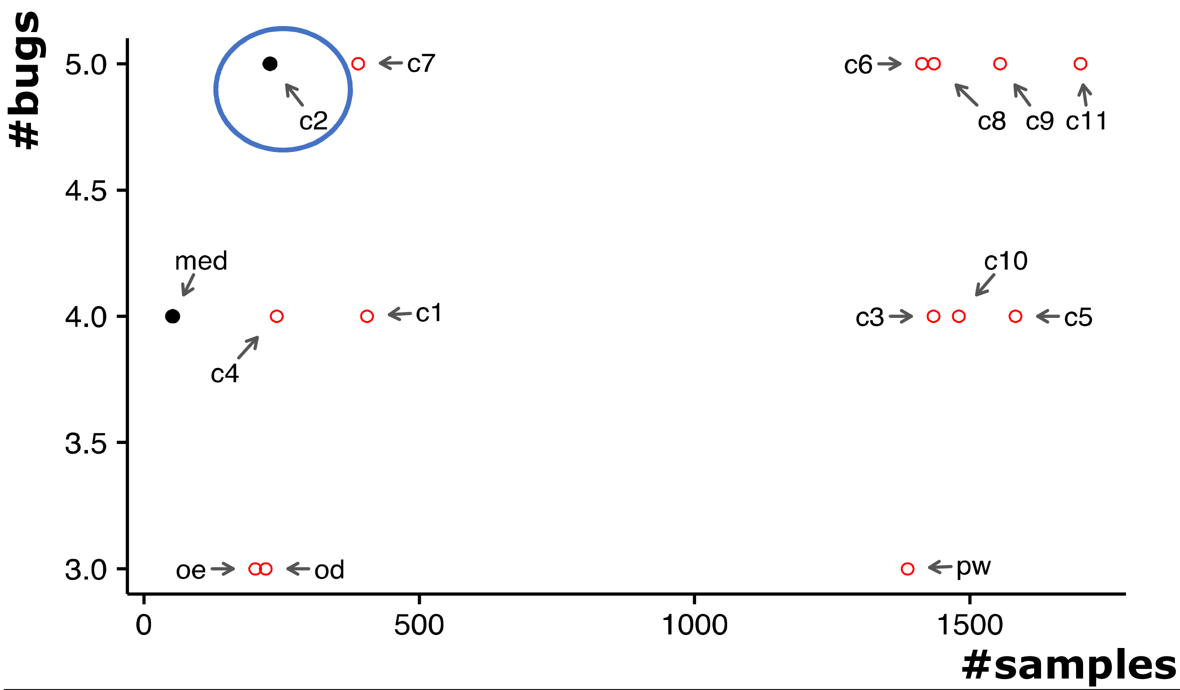
Evaluation

GCC

Version 4.8.2

Findings

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



Bugs found

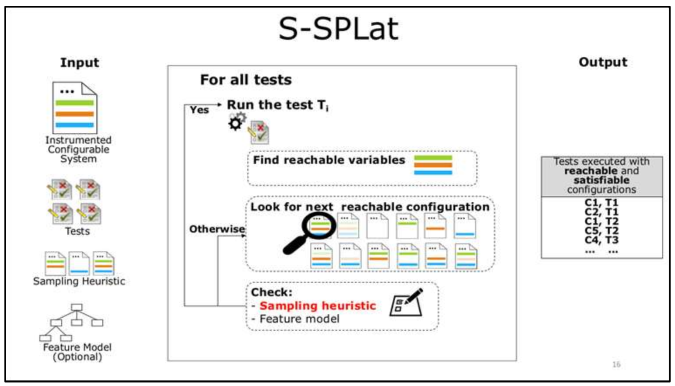
Basic Technique	Crash Id 1	2	3	4	5
med	✓	✓		✓	✓
oe	✓	✓	✓		
od	✓	✓			✓
pw	✓	✓	✓		

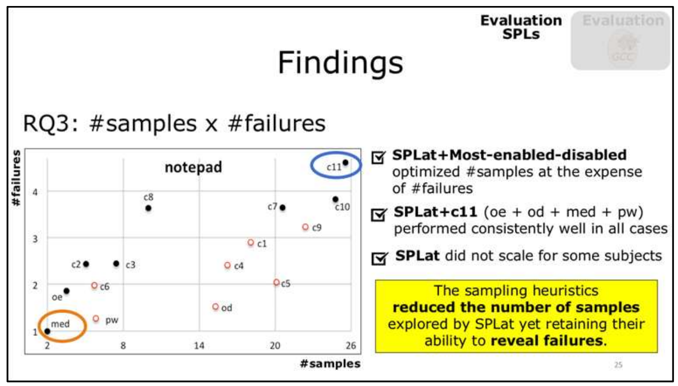
All five bugs were captured.
SPLat+c2(oe+med) found all bugs with a relatively small number of samples.

Lessons Learned

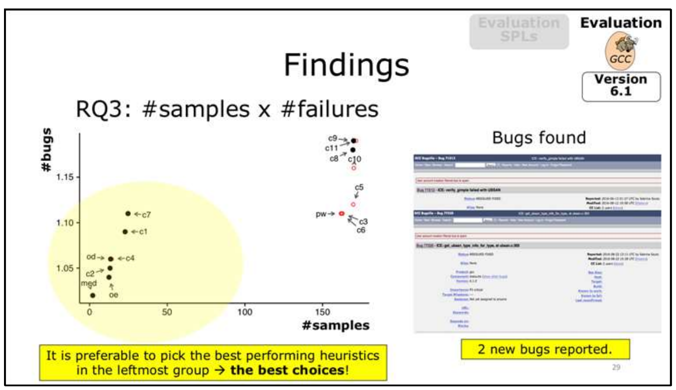
- For SPLs → c11 (oe+od+med+pw)
- For GCC → c2(oe+med)
- For SPLs and GCC → c7 (oe+od+med)
    - [ICSE 2016] **A comparison of 10 sampling algorithms for configurable systems**.
- Combine different simple heuristics
- Avoid heuristics with a large number of requirements

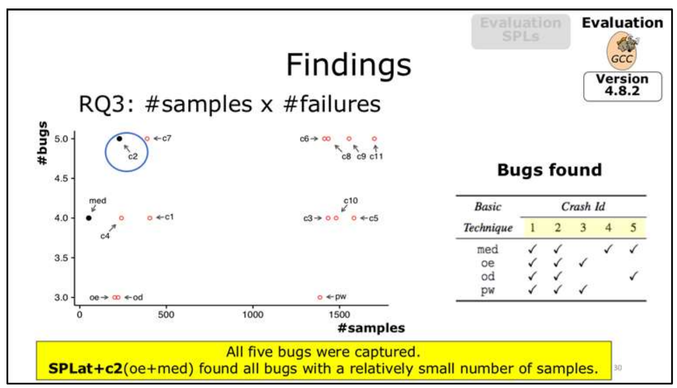
- **S-SPLat found a good balance between bugs and samples**
  - The sampling heuristics helped to reduced the number of samples explored by SPLat without loss the ability to find bugs
- **S-SPLat could deal with scalability**
  - It revealed bugs in potentially large configuration spaces





https://sabrinadfs.github.io/s-splat/



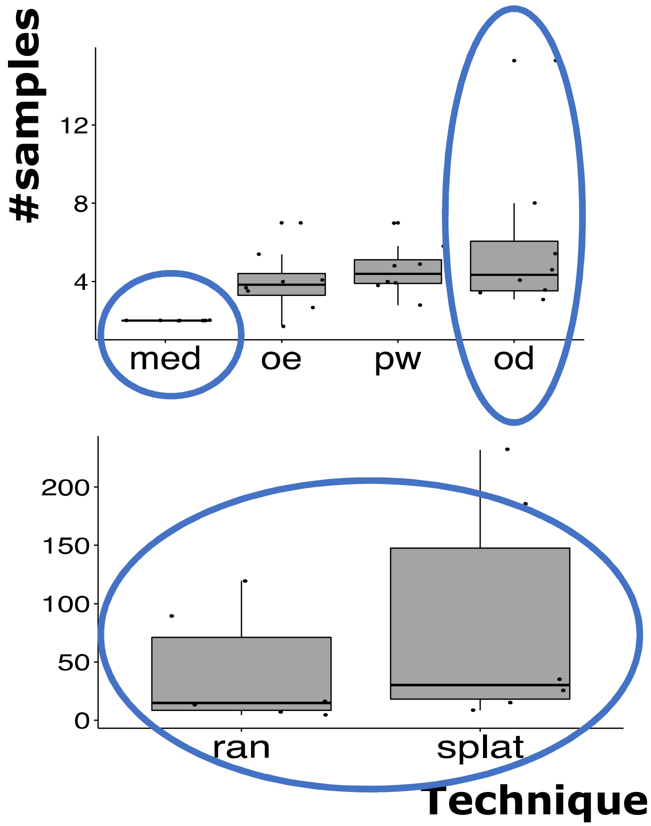


sabrinadfs@gmail.com

BACKUP SLIDES

Evaluation SPLs | Evaluation GCC

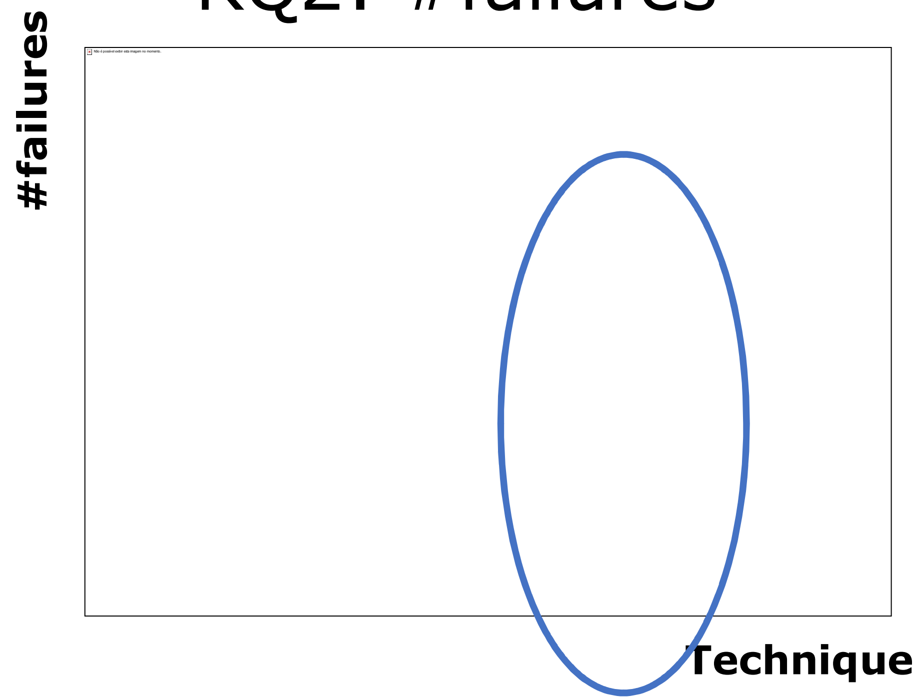
RQ1: #samples



SPLat and **ran** explored much samples.

med explored the smallest sample sets.
od explored the largest sample sets.

RQ2: #failures

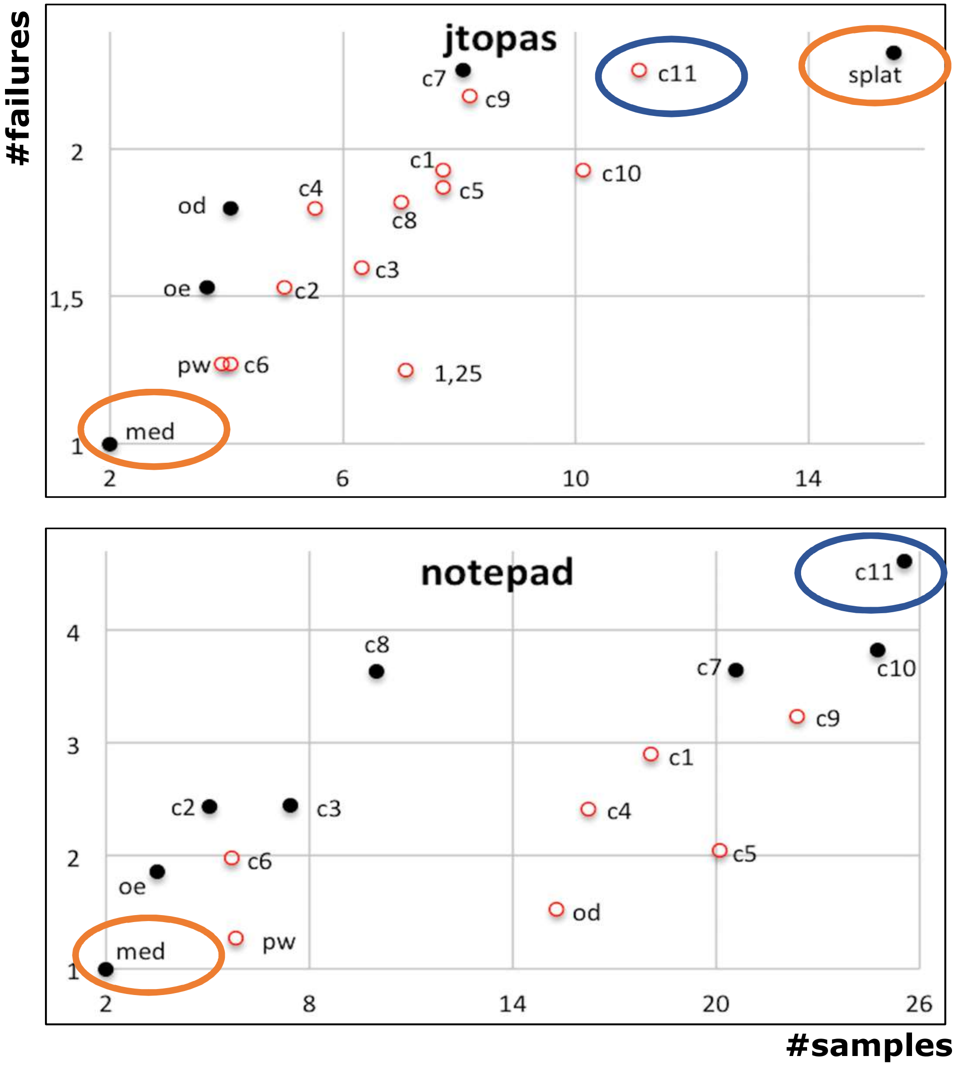


od and **pw** found almost the same number of failures as **splat** but they required much fewer samples.

RQ3: #samples x #failures

Evaluation SPLs

Evaluation GCC



- Combinations of heuristics
  - **oe x od x med x pw**
    - **c1** = oe+od
    - **c2** = oe+med
    - **c3** = oe+pw**…**
    - **c11** = oe+od+med+pw

SPLat and **med** optimize one dimension at the expense of the other.

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**.

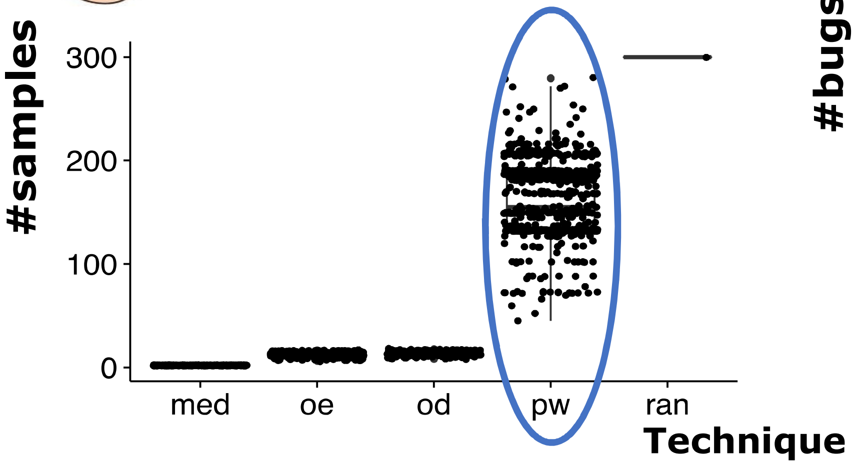
Evaluation SPLs

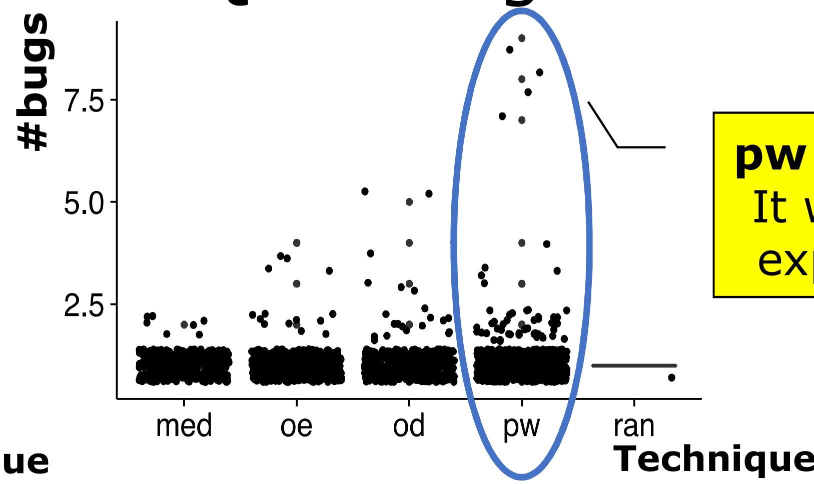
Evaluation GCC

Version 6.1

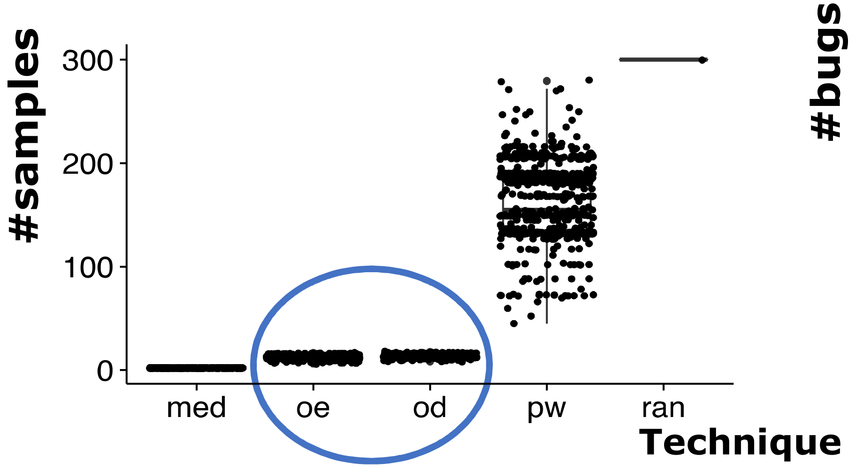
RQ1: #samples

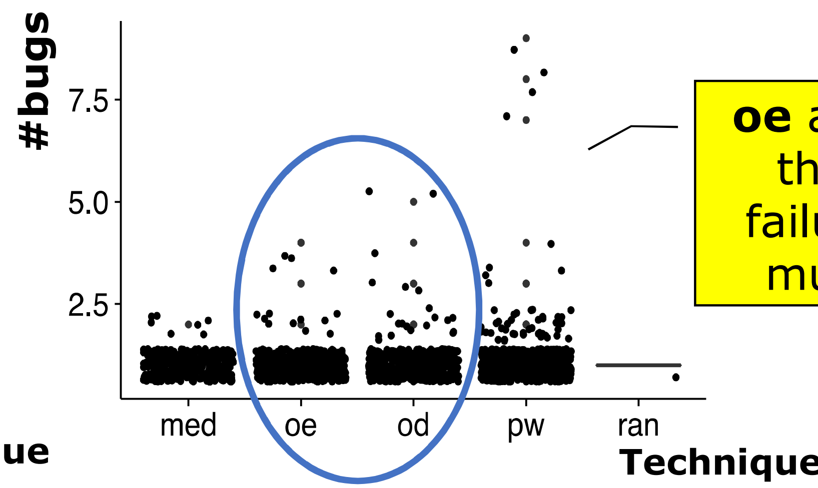
RQ2: #bugs





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





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

Evaluation SPLs

Evaluation GCC

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

Evaluation SPLs **Evaluation GCC**

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