# SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems

...

Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don Batory, Sabrina Souto, Paulo Barros, and Marcelo d'Amorim

slide author names omitted for FERPA compliance

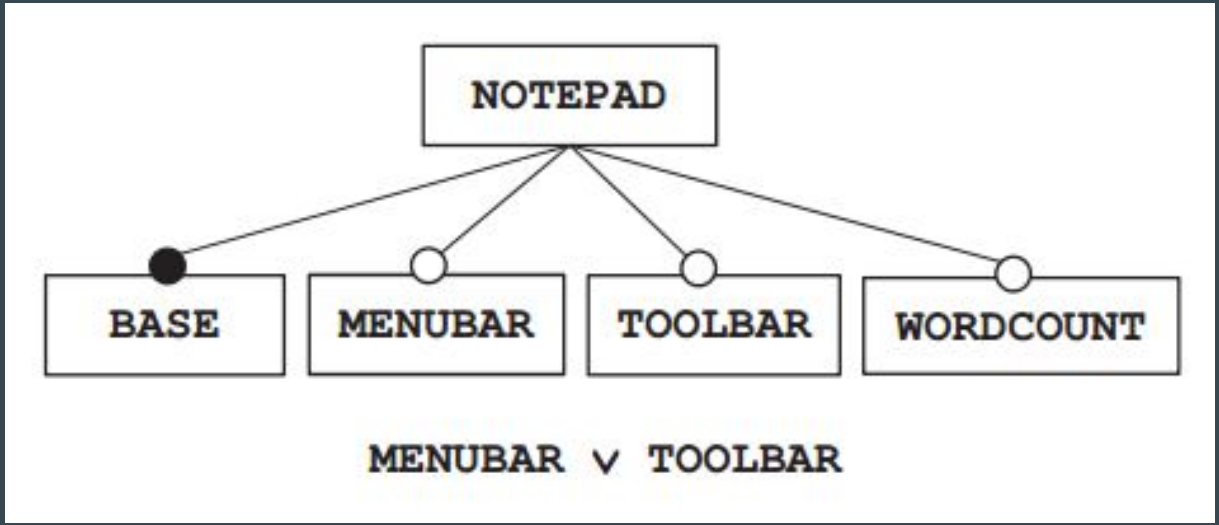
# Terminology

Software Product Line (SPL)

- Specifies a family of programs where each program is defined by a unique combination of features. This work only investigates boolean features-can be present or absent
- “Configuration”, “feature combination”, and “program” are used interchangeably

# Simple Example

For an example of an SPL consider a “Notepad” product line



# Simple Example

```
1  class Notepad extends JFrame {
2   Notepad() {
3    getContentPane().add(newJTextArea());
4   }
5
6   void createToolBar() {
7    if(TOOLBAR) {
8     JToolBar toolBar = new JToolBar();
9     getContentPane().add
10                    ("North", toolBar);
11    if(WORDCOUNT) {
12     JButton button = new
13             JButton("wordcount.gif");
14     toolBar.add(button);
15    }
16   }
17  }
18
19  void createMenuBar() {
20   if(MENUBAR) {
21    JMenuBar menuBar = new JMenuBar();
22    setJMenuBar(menuBar);
23    if(WORDCOUNT) {
24     JMenu menu = new
25                    JMenu("Word Count");
26     menuBar.add(menu);
27    }
28   }
29  }
30 }
```

(a) Code

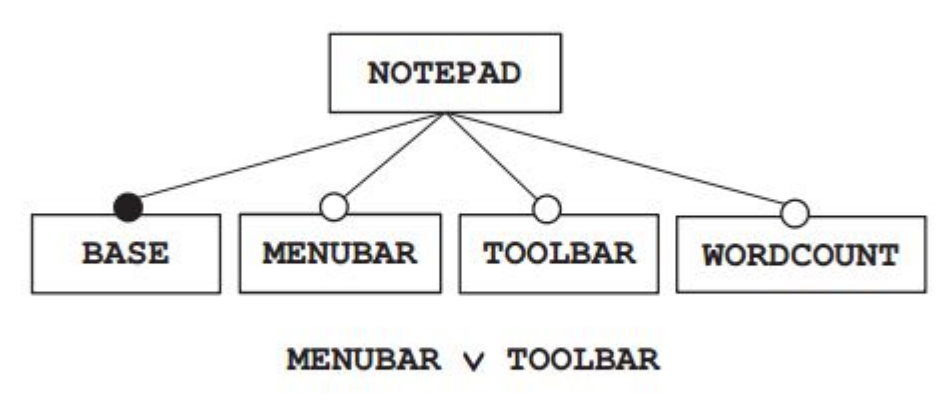
```
1  public void test() {
2   Notepad n = new Notepad();
3   n.createToolBar();
4
5   // Automated GUI testing
6   FrameFixture f = newFixture(n);
7   f.show();
8   String text = "Hello";
9   f.textBox().enterText(text);
10  f.textBox().requireText(text);
11  f.cleanUp();
12 }
```

(b) Test

# Simple Example

We need to ensure the functionality over all configurations

How many configurations are there?

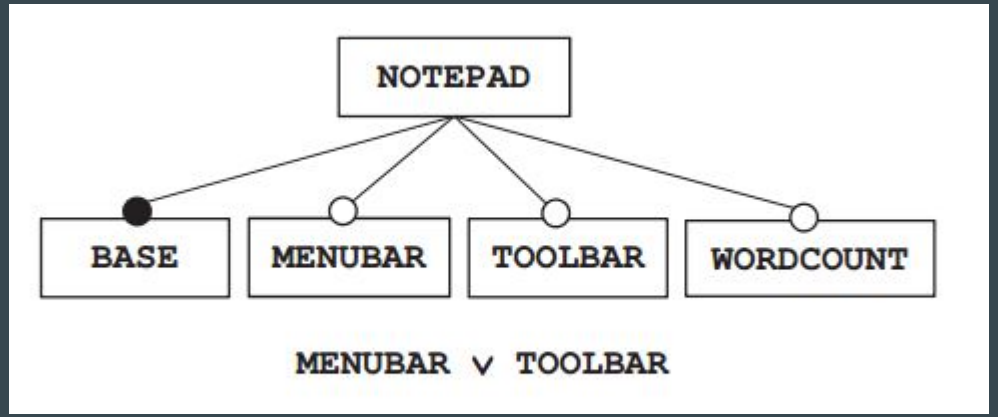


# Simple Example

8 possible combinations of optional features

MTW =

| |
|---|
| 000 |
| 001 |
| 010 |
| 011 |
| 100 |
| 101 |
| 110 |
| 111 |



# Testing SPLs

- Testing SPL is expensive as it requires running each test against combinatorial number of configurations

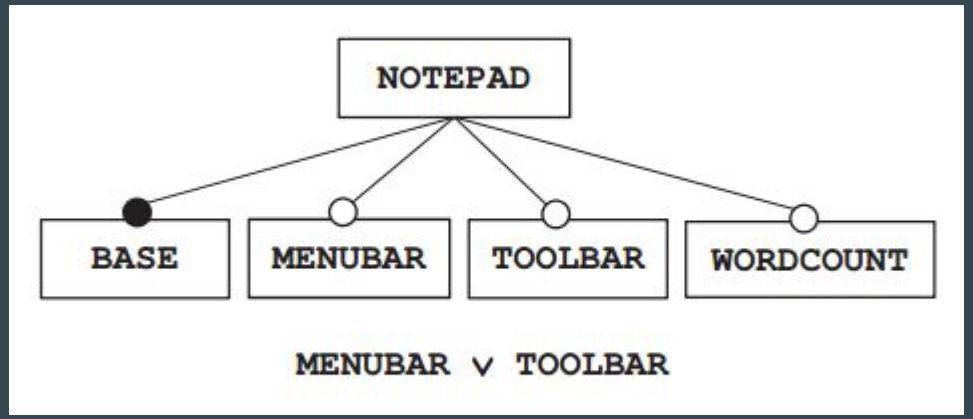
    5 boolean features, gives you 32 configurations, 170 yields ~150000000000000000000000000000000000000000000000000 combinations

# Simple Example

Do we need to test all 8 combinations?

MTW =

| |
|---|
| 000 |
| 001 |
| 010 |
| 011 |
| 100 |
| 101 |
| 110 |
| 111 |

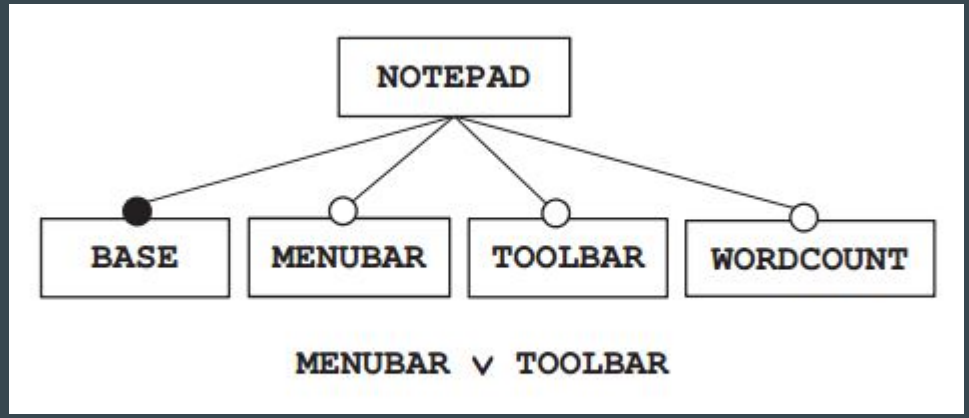


# Simple Example

Just given the model, 2 are invalid

MTW =

| |
|---|
| 000 |
| 001 |
| 010 |
| 011 |
| 100 |
| 101 |
| 110 |
| 111 |

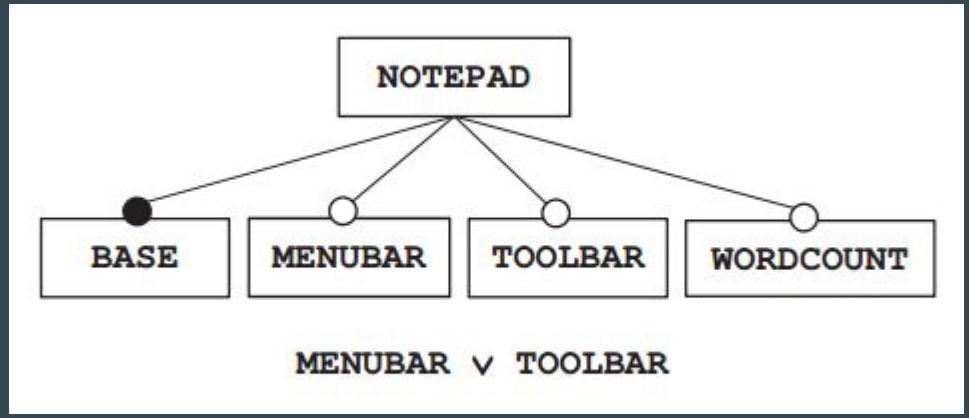


# Simple Example

Just given the model, 2 are invalid. Can we do better?

MTW =

| |
|---|
| 000 |
| 001 |
| 010 |
| 011 |
| 100 |
| 101 |
| 110 |
| 111 |



# Simple Example

Consider MTW 100 and 101

They give the same trace!

```
1  public void test() {
2   Notepad n = new Notepad();
3   n.createToolBar();
4
5   // Automated GUI testing
6   FrameFixture f = newFixture(n);
7   f.show();
8   String text = "Hello";
9   f.textBox().enterText(text);
10  f.textBox().requireText(text);
11  f.cleanUp();
12 }
```

(b) Test

```
1  class Notepad extends JFrame {
2   Notepad() {
3    getContentPane().add(newJTextArea());
4   }
5
6   void createToolBar() {
7    if(TOOLBAR) {
8     JToolBar toolBar = new JToolBar();
9     getContentPane().add
10                    ("North", toolBar);
11    if(WORDCOUNT) {
12     JButton button = new
13             JButton("wordcount.gif");
14    toolBar.add(button);
15    }
16   }
17  }
18
19  void createMenuBar() {
20   if(MENUBAR) {
21    JMenuBar menuBar = new JMenuBar();
22    setJMenuBar(menuBar);
23    if(WORDCOUNT) {
24     JMenu menu = new
25                  JMenu("Word Count");
26    menuBar.add(menu);
27    }
28   }
29  }
30 }
```

(a) Code

# Key Idea

Tests are often independent of many of the configurations!

Tests often focus on small part of the system

Configurations not required can be pruned from the execution

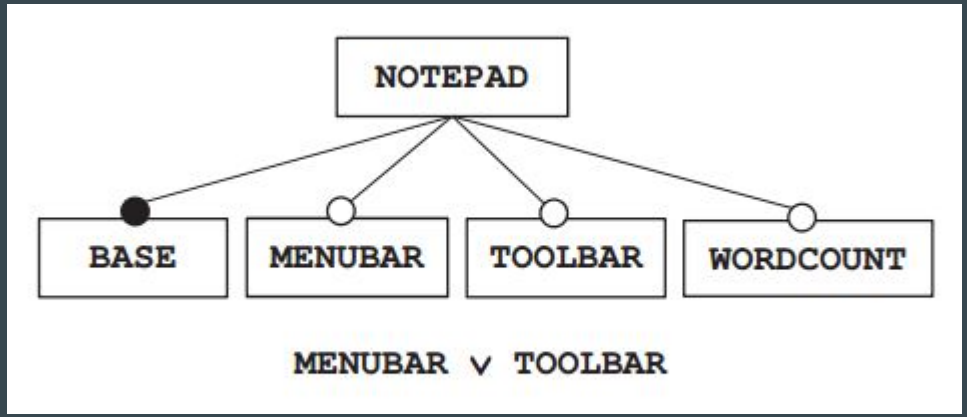
Configurations to run can be determined during testing by monitoring accesses to configuration variables

# Simple Example

Just given the model, 2 are invalid, but SPLat can reduce further

MTW =

| |
|---|
| 000 |
| 001 |
| 010 |
| 011 |
| 100 |
| 101 |
| 110 |
| 111 |



# SPLat algorithm

```
Given a test and feature model, instrument features to observe reads

do{
    Execute test.
    if feature is read push feature on stack and record assignment to state
    while the stack is not empty {
        look at top feature
        if this feature is true in the state, (it has been explored)
            pop feature from the stack and set to false
        else
            put feature into the state as true
            if state is valid for model, break
    }
}while the stack is not empty
```

# SPLat algorithm example run

Load notepad feature model, 3 optional features to explore. Start at MTW 000

TOOLBAR is read first, so pushed onto the stack. When false, no other features are read before the test ends, so we cover MTW -0- where “-” represents “don’t care”

00- are invalid given the feature model, so this one execution covers only 10- configurations. (Even though WORDCOUNT doesn’t matter, we’ll assign it 0 because the features need concrete values so MTW is 100)

# SPLat algorithm example run

Next sets TOOLBAR to true as it is satisfiable, this covers -10. (Again, we'll assign M 0 because the features need concrete values so MTW is 010). WORDCOUNT is encountered, so it is pushed onto the stack

Next sets WORDCOUNT true, this covers -11 (sets M to 0 to execute, MTW is 011).

WORDCOUNT is popped off stack because all values have been explored, and TOOLBAR is popped off as well for the same reason

So three executions 100, 010, and 011 are executed and cover all 6 valid configurations.

# Nuances

Needs reset function to reset test conditions between runs (In our example just restart JVM)

Might need to reset database conditions etc. In evaluation reset functionality was already in place at GROUPON

Could optimize to synchronize between the exploration state and the feature model, which would increase speed

# Research Questions

Systematically testing SPL programs is expensive

How can this be more efficient?

# How can we...

Reduce number of executions?

Reduce overhead?

Improve scalability

# Contributions

Lightweight analysis of configurable programs

- ➔ Lightweight monitoring to speed up test execution
- ➔ Easily implemented in different run-time environments

# Contributions - Implementation


Java


Ruby on Rails

# Contributions - Evaluation

Evaluate SPLat on 10 Java SPLs

# Contributions - Evaluation

Identifies relevant configurations with a low overhead

# Contributions - Evaluation

Apply SPLat on 171KLOC in Ruby on Rails

# Contributions - Evaluation

170 configuration variables

19K tests

231KLOC in Ruby on Rails

# Evaluation

Ten configurable Java programs were converted into Subject SPLs

| *SPL* | *Features* | *Confs* | *LOC* |
|---|---|---|---|
| 101Companies | 11 | 192 | 2,059 |
| Elevator | 5 | 20 | 1,046 |
| Email | 8 | 40 | 1,233 |
| GPL | 14 | 73 | 1,713 |
| JTopas | 5 | 32 | 2,031 |
| MinePump | 6 | 64 | 580 |
| Notepad | 23 | 144 | 2.074 |
| Prevayler | 5 | 32 | 2,844 |
| Sudoku | 6 | 20 | 853 |
| XStream | 7 | 128 | 14,480 |

# Tests done on subjects

`LOW` : optimistic

`MED` : average

`HIGH` : pessimistic

# Comparable Techniques

NewJVM - spawns a new JVM for each distinct run. Each test run executes **one** valid configuration

ReuseJVM - uses the same JVM across several runs. Reset function is required.

SRA (Static Reachable Analysis) - performs reachability analysis, control-flow and data-flow analyses to statistically figure out which configurations are reachable from a given test

# RQ1: Efficiency

How does SPLat's efficiency compare with alternative techniques for analyzing SPL tests?

- Tests show that reusing JVM is about 50% faster than starting up a new JVM every time
- In comparison to SRA
  - Uses less configurations because SRA is conservative
  - SPL Overhead < SRA Overhead (by a lot)
  - SPL IdealTime < SRA Time

# RQ2: Overhead

What is the overhead of SPLat?

| SPLat | | | | | | Static Reachability (SRA) | | |
|---|---|---|---|---|---|---|---|---|
| *Confs* | | *SPLatTime* | | *IdealTime* | *Overhead* | *Confs* | *Overhead* | *Time* |
| **101Companies (192 configs)** | | | | | | | | |
| 32 | (16%) | 1.64 | (77%) | 0.72 | 0.92 (127%) | 96 | 84.04 | 1.28 |
| 160 | (83%) | 6.84 | (175%) | 3.58 | 3.26 (91%) | 192 | 82.54 | 3.99 |
| 176 | (91%) | 47.6 | (105%) | 41.59 | 6.01 (14%) | 192 | 81.93 | 45.16 |
| **Notepad (144 configs)** | | | | | | | | |
| 2 | (1%) | 3.06 | (2%) | 2.45 | 0.61 (24%) | 144 | 80.40 | 135.47 |
| 96 | (66%) | 104.95 | (67%) | 104.91 | 0.04 (0%) | 144 | 80.62 | 156.35 |
| 144 | (100%) | 153.11 | (99%) | 152.16 | 0.94 (0%) | 144 | 81.29 | 151.94 |

Large overhead for short-running tests
Small overhead for long-running tests

# RQ2: Overhead

| SPLat | | | | | | | Static Reachability (SRA) | | |
|---|---|---|---|---|---|---|---|---|---|
| *Confs* | | *SPLatTime* | | *IdealTime* | *Overhead* | | *Confs* | *Overhead* | *Time* |
| **JTopas (32 configs)** | | | | | | | | | |
| 8 | (25%) | 6.29 | (37%) | 4.49 | 1.80 | (40%) | 32 | 86.87 | 16.44 |
| 16 | (50%) | 13.16 | (70%) | 9.71 | 3.46 | (35%) | 32 | 86.87 | 18.70 |
| 32 | (100%) | 25.31 | (133%) | 18.43 | 6.88 | (37%) | 32 | 86.87 | 18.48 |
| **MinePump (64 configs)** | | | | | | | | | |
| 9 | (14%) | 3.65 | (48%) | 1.90 | 1.75 | (91%) | 64 | 22.69 | 7.49 |
| 24 | (37%) | 10.43 | (70%) | 6.26 | 4.17 | (66%) | 64 | 22.38 | 15.35 |
| 48 | (75%) | 37.80 | (657%) | 4.81 | 32.99 | (685%) | 64 | 22.18 | 5.77 |

JTopas: Feature variables are accessed many times because they are accessed within the tokenizing loop

MinePump: Test subject is small (580 LOC)

# RQ3: Scalability


Does SPLat scale to real code?

- Groupon PWA is the codebase that powers the whole website
- Frameworks for testing: Rspec, Cucumber, Selenium, and Jasmine
- SPLat was implemented to Ruby on Rails to apply it to Groupon PWA
- Reset functions already implemented
- Highly configurable (170 feature variables)
- Set limit to configurations to 16

Does scale to real code. The implementation effort and the number of configurations for SPLat in real tests is relatively low.

# RQ3: Scalability

Most real tests indeed cover a small number of configurations.

Reachable Configurations

| *Configs* | *Tests* | *Configs* | *Tests* |
|---|---|---|---|
| 1 | 11,711 | 2 | 1,757 |
| 3 | 332 | 4 | 882 |
| 5 | 413 | 6 | 113 |
| 7 | 19 | 8 | 902 |
| 9 | 207 | 10 | 120 |
| 11 | 29 | 12 | 126 |
| 13 | 6 | 14 | 32 |
| 15 | 10 | 16 | 349 |
| 17 | 2,695 | - | - |

Accessed Features

| *Vars* | *Tests* | *Vars* | *Tests* | *Vars* | *Tests* |
|---|---|---|---|---|---|
| 0 | 11,711 | 1 | 1,757 | 2 | 1,148 |
| 3 | 1,383 | 4 | 705 | 5 | 389 |
| 6 | 466 | 7 | 323 | 8 | 425 |
| 9 | 266 | 10 | 140 | 11 | 86 |
| 12 | 80 | 13 | 34 | 14 | 28 |
| 15 | 54 | 16 | 62 | 17 | 1 |
| 19 | 14 | 20 | 260 | 21 | 109 |
| 22 | 45 | 23 | 19 | 24 | 22 |
| 25 | 9 | 26 | 2 | 27 | 14 |
| 28 | 17 | 29 | 6 | 30 | 8 |
| 31 | 24 | 32 | 6 | 33 | 14 |
| 34 | 31 | 35 | 11 | 36 | 15 |
| 37 | 8 | 38 | 2 | 39 | 2 |
| 40 | 3 | 42 | 2 | 43 | 2 |

# Discussion Question

How can effective can applying this in industry be?

# Discussion Question

How can effective can applying this in industry be?

Groupon example shows the technique scales to large SPLs, however the results do not take into account the cost of writing the reset function
(as one already existed in study)

# Discussion Question

How difficult is it to implement SPLat compared to current techniques of testing SPLs

# Discussion Question

How difficult is it to implement SPLat compared to current techniques of testing SPLs

The authors provided two implementations, a Java one that built on top of Korat that integrated a SAT solver, and a Ruby on Rails implementation that didn't use a feature model or SAT solver (treated all configurations as valid).

# Discussion Question

Can we use SPLat on SPLs with more than just boolean features?

# Discussion Question

Can we use SPLat on SPLs with more than just boolean features?

Could represent ternary as boolean (just more possible configurations states)

# Discussion Question

Does this solve the SAT problem?

# Discussion Question

Does this solve the SAT problem?

No, uses heuristic solver, not deterministic polynomial time algorithm

# Discussion Question

How is SPLat different from Korat?

# Discussion Question

How is SPLat different from Korat?

Korat encodes a precondition for running the configurable system, which must be accounted for.