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

Kim, Chang Hwan Peter, et al.

presenter name(s) removed for FERPA considerations

Introduction

Same phone, different models

They share some common features and some features vary:

- Screen Size
- Colors
- RAM
- Camera
- so on..

Benefits:

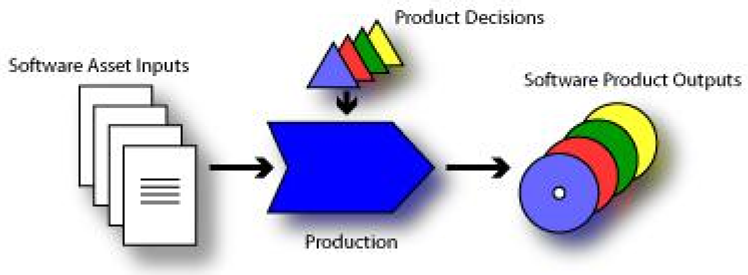
- Satisfies the specific needs of a particular market segment
- Decrease production costs
- Decrease time to market
- Increase quality
- so on..

Introduction

Let’s think of software the same way!

Software Product Line: Family of programs where each program is defined by a unique combination of features.

Idea is to manufacture software from reusable parts



Introduction

- **Problem with SPLs?**

  Testing is expensive: running each test against all possible configurations.

170 boolean configuration variables

Can be deployed in 2^{170} different configurations!!

Introduction

- **Current Solution**

  Test is often independent of many configuration variables and need not be run against every configuration.

  Such irrelevant configurations can be pruned from execution.

  Current techniques:

  - *Sampling* (Random or sophisticated selection of configurations)

    Chance of missing out on bugs!

  - *Exhaustive Exploration* (Static or Heavyweight Dynamic Analysis)

    Too much execution overhead!

Main Contribution

SPLat introduces a,

new *light-weight* technique for analysis of configurable programs

Key Idea

- Some features are never encountered, no matter how the test is executed.
- Combinations of such unreachable features lead to the same *trace.*

For any test,

SPLat runs only the configurations that have a unique trace

Example

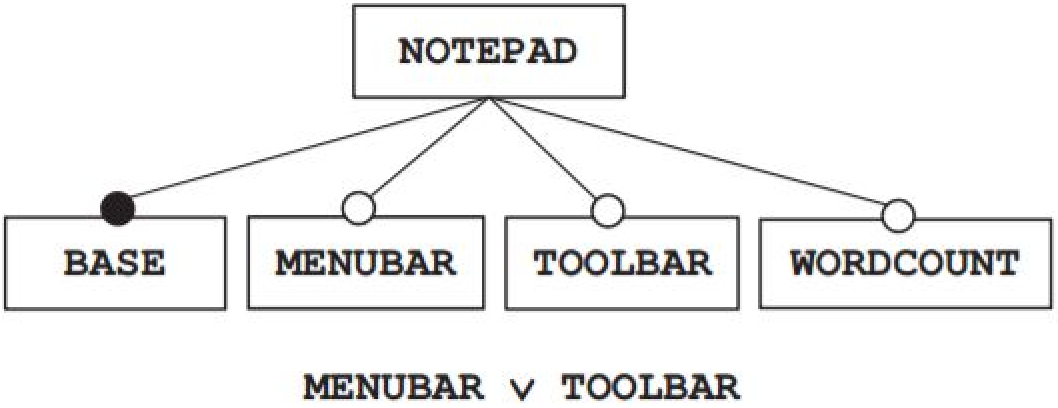
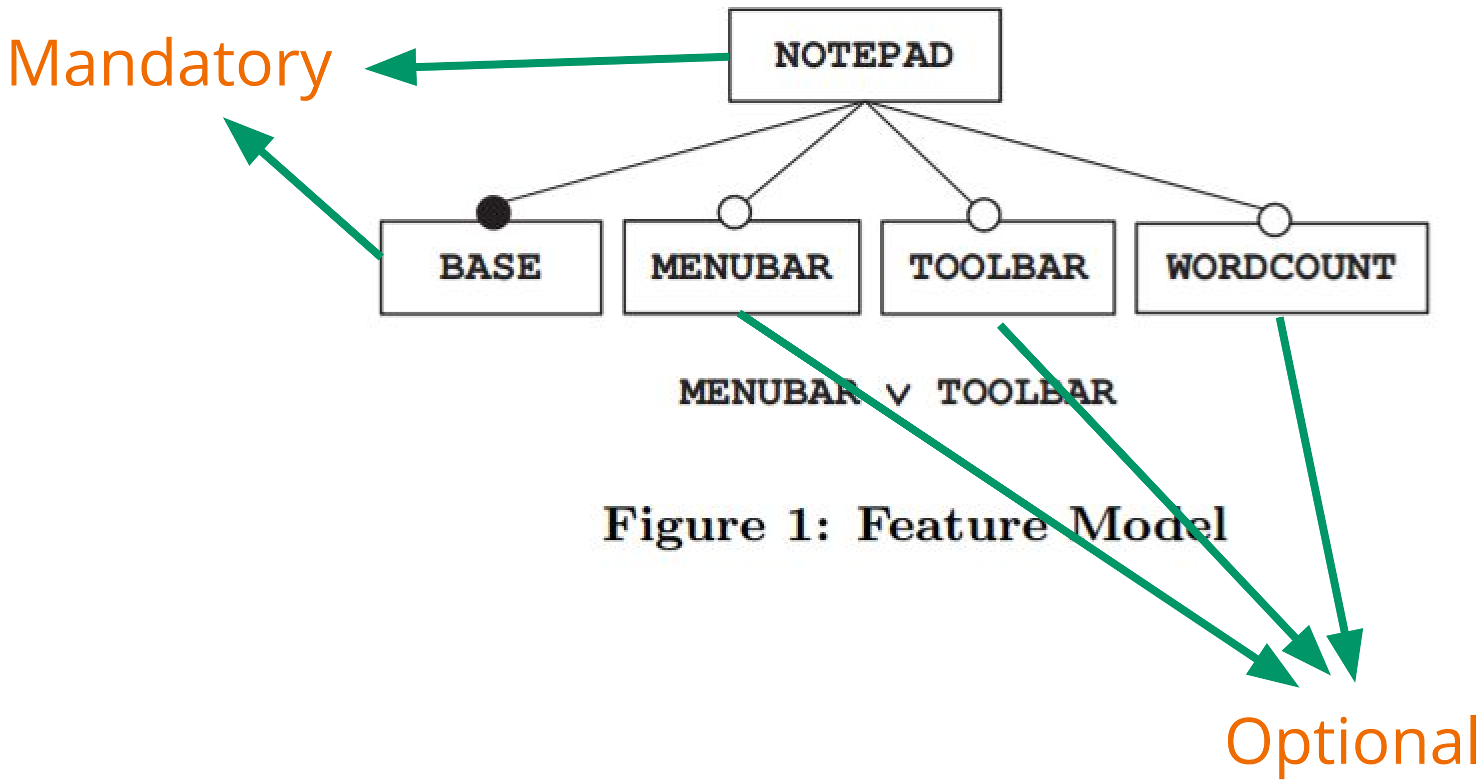


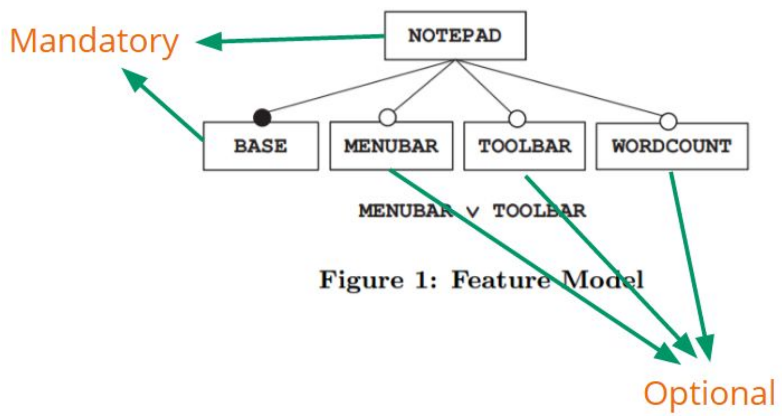
Figure 1: Feature Model

Example



Constraint: Must have a MENUBAR or TOOLBAR

Example



- Mandatory features are always *true*
- Optional features may be set to *true* or *false*
- Configuration = assignment of values to all feature variables
- Constraint
  - Must have a MENUBAR or TOOLBAR

Configuration is valid iff it satifies all constraints the feature model expresses

Example

```
public void test() {
 Notepad n = new Notepad();
 n.createToolBar();

 // Automated GUI testing
 FrameFixture f = newFixture(n);
 f.show();
 String text = "Hello";
 f.textBox().enterText(text);
 f.textBox().requireText(text);
 f.cleanUp();
}
```

(b) Test

```
class Notepad extends JFrame {
 Notepad() {
  getContentPane().add(newJTextArea());
 }

 void createToolBar() {
  if(TOOLBAR) {
   JToolBar toolBar = new JToolBar();
   getContentPane().add
                 ("North", toolBar);
   if(WORDCOUNT) {
    JButton button = new
           JButton("wordcount.gif");
    toolBar.add(button);
   }
  }
 }

 void createMenuBar() {
  if(MENUBAR) {
   JMenuBar menuBar = new JMenuBar();
   setJMenuBar(menuBar);
   if(WORDCOUNT) {
    JMenu menu = new
                JMenu("Word Count");
    menuBar.add(menu);
   }
  }
 }
}
```

(a) Code

Example

Without SPLat :

3 Optional Variables = *8 configurations*

M: MENUBAR

T: TOOLBAR

W: WORDCOUNT

MTW = 000
MTW = 001

MTW = 100
MTW = 101

MTW = 110
MTW = 010

MTW = 011
MTW = 111

Example

With SPLat :

3 Optional Variables = ***6** configurations*

M: MENUBAR

T: TOOLBAR

W: WORDCOUNT

Prune invalid configurations

MTW = 000 ✖

MTW = 001 ✖

Both M and T false, constraint not satisfied

MTW = 100

MTW = 101

MTW = 110

MTW = 010

MTW = 011

MTW = 111

Constraint: Must have a MENUBAR or TOOLBAR

Example

With SPLat :

3 Optional Variables = ***5** configurations*

M: MENUBAR

T: TOOLBAR

W: WORDCOUNT

Remove valid configurations that are unnecessary

MTW = 000 ✖
MTW = 001 ✖

Both M and T false, constraint not satisfied

MTW = 100 ✓
MTW = 101 ✖

Non-unique trace, same as 100

MTW = 110
MTW = 010

MTW = 011
MTW = 111

```
public void test() {
 Notepad n = new Notepad();
 n.createToolBar();

 // Automated GUI testing
 FrameFixture f = newFixture(n);
 f.show();
 String text = "Hello";
 f.textBox().enterText(text);
 f.textBox().requireText(text);
 f.cleanUp();
}
```

Test only calls *createToolBar()* which is empty in both configurations.

Example

With SPLat :

3 Optional Variables = ~~*8 configurations*~~ ***3 configurations!***

M: MENUBAR

T: TOOLBAR

W: WORDCOUNT

MTW = 000 ✖ Both M and T false, constraint not satisfied
MTW = 001 ✖

MTW = 100 ✔
MTW = 101 ✖ Non-unique trace, same as 100

MTW = 010 ✔
MTW = 110 ✖

MTW = 011 ✔
MTW = 111 ✖

Technique

Main Algorithm Overview:

1) Run **initial instance** of a test for the SPL

2) **Set features**
    - Mandatory features - true
    - Optional features - true or false. Defaulted to false

3) **Explore**, **backtrack**, and **explore more**
    - New features are pushed onto a stack as they are encountered

4) **Prune invalid** configurations until exploration is done

Technique

M: MENUBAR — 0 : False

T: TOOLBAR — 1 : True

W: WORDCOUNT — - : “don’t care”

Applied Example - Notepad:

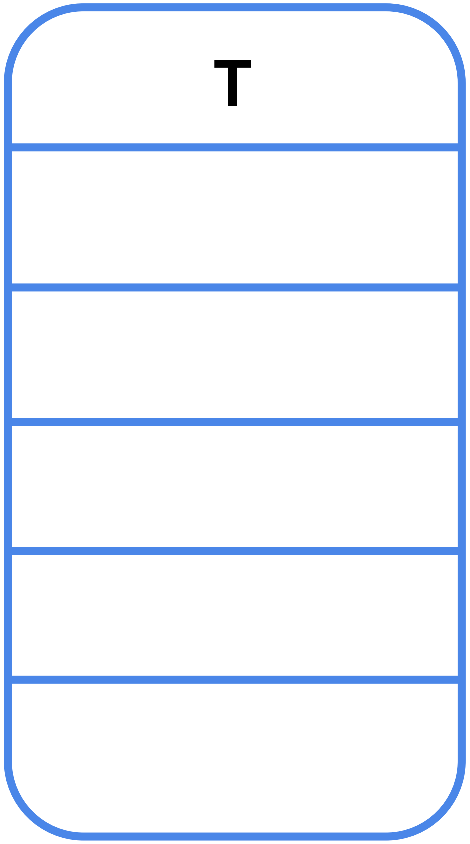
1) **Define a stack** to hold unexplored features

2) Run an **initial test** and gather set of partial assignments

   T is encountered first, push it on to the stack

   Stack: [T]

   Set T=0 → MTW = -0-



Technique

M: MENUBAR 0 : False
T: TOOLBAR 1 : True
W: WORDCOUNT - : “don’t care”

Applied Example - Notepad:

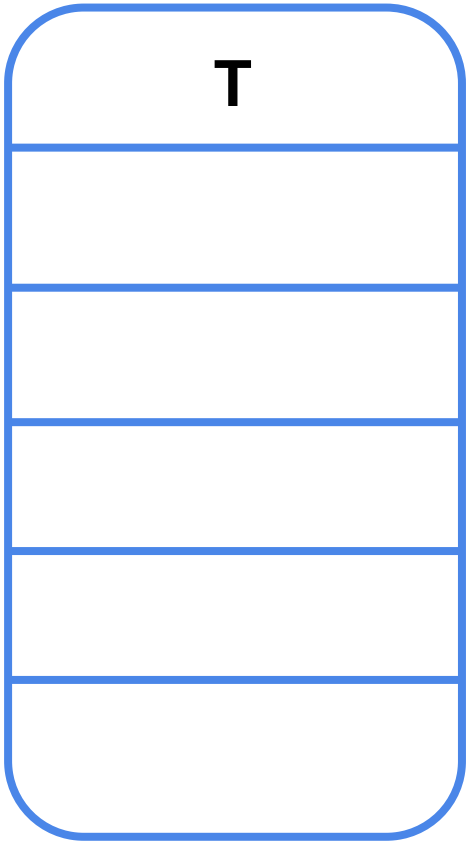
3) Explore **initial set of assignments**

Explore MTW=-0-:

Recognize that MTW=00- is invalid →

Two combinations eliminated!

Assert MTW=10-: valid



Technique

M: MENUBAR 0 : False

T: TOOLBAR 1 : True

W: WORDCOUNT - : “don’t care”

Applied Example - Notepad:

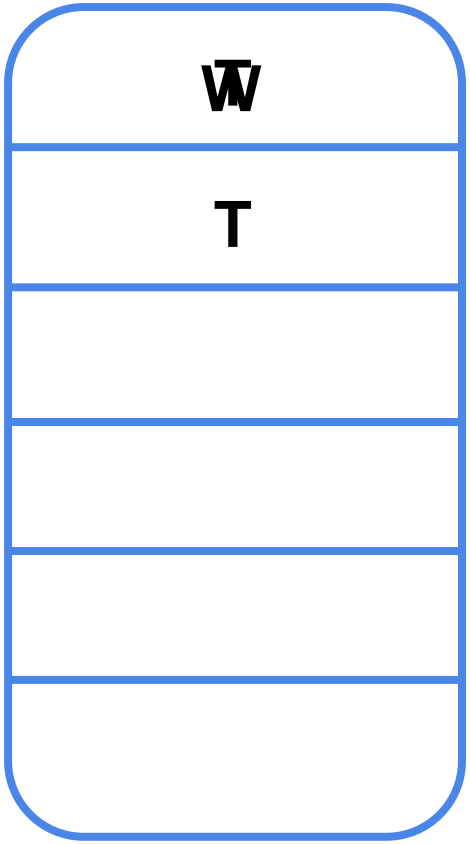
4) **Bubble up** and explore alternate path of configurations

    Set T=1, explore path (MTW=-1-):

        Next, we encounter W. Push W. (Stack: [W, T])

            MTW=-10: valid

            MTW=-11: valid



Technique

M: MENUBAR 0 : False

T: TOOLBAR 1 : True

W: WORDCOUNT - : “don’t care”

Applied Example - Notepad:

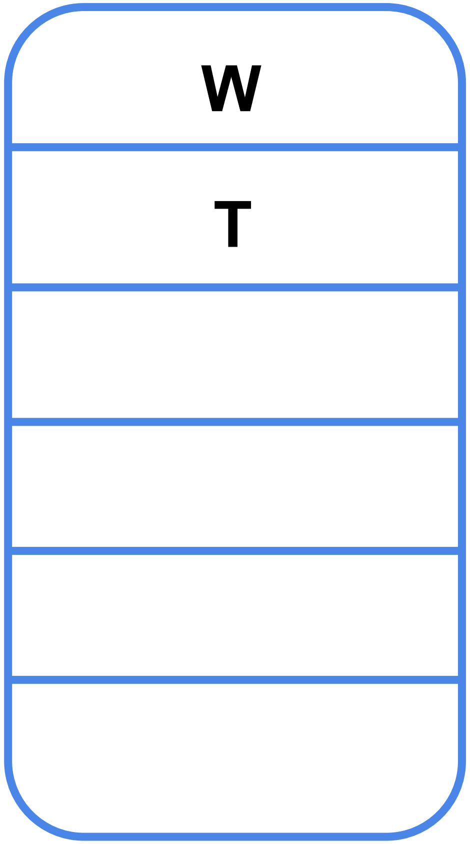
5) **Pop out** fully explored features

W & T have been fully explored at this point

Pop W.

Pop T.

Stack is empty, end SPLat



Technique

M: MENUBAR — 0 : False

T: TOOLBAR — 1 : True

W: WORDCOUNT — - : “don’t care”

Applied Example - Notepad:

6) **Collect results**

MTW=10-, MTW=-10, and MTW=-11 are all covered in 3 test executions

Thus, 6 feasible combinations were fully explored in 3 configurations!

Evaluation

- **RQ1:** Compare with other methods
- **RQ2:** Overhead?
- **RQ3:** Scale to industry code?

RQ1 & RQ2: Experiment on SPL

- 10 software product lines (SPLs) as test subjects

Table 1: 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

- Comare with:
- **Naive approach:** run on *all* valid configurations
  - **NewJVM**: spawn a new JVM for each test run
  - **ReuseJVM**: require an explicit reset function
- **Static approach:**
  - **SRA** (Static Reachablility Analysis): determine reachable configurations by static analysis

JVM: Java Virtual Machine
Required for running Java
Consumes time to create

No comparison with other dynamic approaches!

Table 2: Experimental Results for Various Techniques

	All Valid			SPLat							Static Reachability (SRA)		
Test	*NewJVM*	*ReuseJVM*		*Confs*		*SPLatTime*		*IdealTime*	*Overhead*		*Confs*	*Overhead*	*Time*
101Companies (192 configs)													
LOW	35.46	2.13	(6%)	32	(16%)	1.64	(77%)	0.72	0.92	(127%)	96	84.04	1.28
MED	49.37	3.90	(7%)	160	(83%)	6.84	(175%)	3.58	3.26	(91%)	192	82.54	3.99
HIGH	283.69	45.26	(15%)	176	(91%)	47.6	(105%)	41.59	6.01	(14%)	192	81.93	45.16
Elevator (20 configs)													
LOW	10.74	5.17	(48%)	2	(10%)	1.33	(25%)	0.71	0.62	(87%)	2	23.29	0.76
MED	50.97	46.65	(91%)	10	(50%)	23.62	(50%)	23.14	0.48	(2%)	20	23.74	46.17
HIGH	62.57	59.48	(95%)	20	(100%)	60.71	(102%)	59.28	1.43	(2%)	20	24.38	60.43
Email (40 configs)													
LOW	40.63	10.74	(26%)	1	(2%)	1.00	(9%)	0.87	0.13	(14%)	1	23.62	0.87
MED	57.56	48.87	(84%)	30	(75%)	36.99	(75%)	37.14	-0.15	(0%)	40	22.81	49.02
HIGH	58.02	48.93	(84%)	40	(100%)	48.96	(100%)	49.26	-0.31	(0%)	40	23.84	49.16
GPL (73 configs)													
LOW	19.21	2.23	(11%)	6	(8%)	0.79	(35%)	0.29	0.49	(168%)	6	104.97	0.30
MED	190.53	171.62	(90%)	55	(75%)	130.87	(76%)	128.52	2.35	(1%)	55	99.41	128.69
HIGH	314.20	285.89	(90%)	70	(95%)	278.77	(97%)	277.48	1.29	(0%)	73	103.52	286.28
JTopas (32 configs)													
LOW	26.59	16.83	(63%)	8	(25%)	6.29	(37%)	4.49	1.80	(40%)	32	86.87	16.44
MED	29.04	18.55	(63%)	16	(50%)	13.16	(70%)	9.71	3.46	(35%)	32	86.87	18.70
HIGH	28.92	18.93	(65%)	32	(100%)	25.31	(133%)	18.43	6.88	(37%)	32	86.87	18.48
MinePump (64 configs)													
LOW	23.71	7.53	(31%)	9	(14%)	3.65	(48%)	1.90	1.75	(91%)	64	22.69	7.49
MED	59.72	14.78	(24%)	24	(37%)	10.43	(70%)	6.26	4.17	(66%)	64	22.38	15.35
HIGH	13.72	5.75	(41%)	48	(75%)	37.80	(657%)	4.81	32.99	(685%)	64	22.18	5.77
Notepad (144 configs)													
LOW	398.22	135.60	(34%)	2	(1%)	3.06	(2%)	2.45	0.61	(24%)	144	80.40	135.47
MED	418.23	156.27	(37%)	96	(66%)	104.95	(67%)	104.91	0.04	(0%)	144	80.62	156.35
HIGH	419.99	153.39	(36%)	144	(100%)	153.11	(99%)	152.16	0.94	(0%)	144	81.29	151.94
Prevayler (32 configs)													
LOW	65.34	40.23	(61%)	12	(37%)	22.49	(55%)	22.8	-0.31	(-1%)	32	205.54	45.39
MED	121.38	96.50	(79%)	24	(75%)	102.49	(106%)	105.86	-3.37	(-3%)	32	214.67	111.37
HIGH	149.08	120.7	(80%)	32	(100%)	127.17	(105%)	131.37	-4.20	(-3%)	32	290.66	135.61
Sudoku (20 configs)													
LOW	51.11	48.10	(94%)	4	(20%)	42.72	(88%)	24.12	18.6	(77%)	10	31.87	24.28
MED	118.14	105.67	(89%)	10	(50%)	58.31	(55%)	54.16	4.15	(7%)	10	31.75	53.67
HIGH	489.60	334.82	(68%)	20	(100%)	316.47	(94%)	332.36	-15.89	(-4%)	20	31.74	338.48
Xstream (128 configs)													
LOW	111.26	30.04	(27%)	2	(1%)	1.57	(5%)	1.08	0.49	(45%)	2	106.50	1.06
MED	105.10	9.04	(8%)	64	(50%)	5.77	(63%)	5.26	0.51	(9%)	64	109.22	5.14
HIGH	101.66	8.68	(8%)	128	(100%)	9.16	(105%)	8.59	0.57	(6%)	128	105.68	8.74

Table 2: Experimental Results for Various Techniques

Test	All Valid: NewJVM	All Valid: ReuseJVM		SPLat: Confs		SPLat: SPLatTime		SPLat: IdealTime	SPLat: Overhead		Static Reachability (SRA): Confs	Static Reachability (SRA): Overhead	Static Reachability (SRA): Time
101Companies (192 configs)													
LOW	35.46	2.13	(6%)	32	(16%)	1.64	(77%)	0.72	0.92	(127%)	96	84.04	1.28
MED	49.37	3.90	(7%)	160	(83%)	6.84	(175%)	3.58	3.26	(91%)	192	82.54	3.99
HIGH	283.69	45.26	(15%)	176	(91%)	47.6	(105%)	41.59	6.01	(14%)	192	81.93	45.16
Elevator (20 configs)													
LOW	10.74	5.17	(48%)	2	(10%)	1.33	(25%)	0.71	0.62	(87%)	2	23.29	0.76
MED	50.97	46.65	(91%)	10	(50%)	23.62	(50%)	23.14	0.48	(2%)	20	23.74	46.17
HIGH	62.57	59.48	(95%)	20	(100%)	60.71	(102%)	59.28	1.43	(2%)	20	24.38	60.43
Email (40 configs)													
LOW	40.63	10.74	(26%)	1	(2%)	1.00	(9%)	0.87					
MED	57.56	48.87	(84%)	30	(75%)	36.99	(75%)	37.14					
HIGH	58.02	48.93	(84%)	40	(100%)	48.96	(100%)	49.26					
GPL (73 configs)													
LOW	19.21	2.23	(11%)	6	(8%)	0.79	(35%)	0.29					
MED	190.53	171.62	(90%)	55	(75%)	130.87	(76%)	128.52					
HIGH	314.20	285.89	(90%)	70	(95%)	278.77	(97%)	277.48					
JTopas (32 configs)													
LOW	26.59	16.83	(63%)	8	(25%)	6.29	(37%)	4.49					
MED	29.04	18.55	(63%)	16	(50%)	13.16	(70%)	9.71					
HIGH	28.92	18.93	(65%)	32	(100%)	25.31	(133%)	18.43					
MinePump (64 configs)													
LOW	23.71	7.53	(31%)	9	(14%)	3.65	(48%)	1.90					
MED	59.72	14.78	(24%)	24	(37%)	10.43	(70%)	6.26					
HIGH	13.72	5.75	(41%)	48	(75%)	37.80	(657%)	4.81					
Notepad (144 configs)													
LOW	398.22	135.60	(34%)	2	(1%)	3.06	(2%)	2.45					
MED	418.23	156.27	(37%)	96	(66%)	104.95	(67%)	104.91					
HIGH	419.99	153.39	(36%)	144	(100%)	153.11	(99%)	152.16	0.94	(0%)	144	81.29	151.94
Prevayler (32 configs)													
LOW	65.34	40.23	(61%)	12	(37%)	22.49	(55%)	22.8	-0.31	(-1%)	32	205.54	45.39
MED	121.38	96.50	(79%)	24	(75%)	102.49	(106%)	105.86	-3.37	(-3%)	32	214.67	111.37
HIGH	149.08	120.7	(80%)	32	(100%)	127.17	(105%)	131.37	-4.20	(-3%)	32	290.66	135.61
Sudoku (20 configs)													
LOW	51.11	48.10	(94%)	4	(20%)	42.72	(88%)	24.12	18.6	(77%)	10	31.87	24.28
MED	118.14	105.67	(89%)	10	(50%)	58.31	(55%)	54.16	4.15	(7%)	10	31.75	53.67
HIGH	489.60	334.82	(68%)	20	(100%)	316.47	(94%)	332.36	-15.89	(-4%)	20	31.74	338.48
Xstream (128 configs)													
LOW	111.26	30.04	(27%)	2	(1%)	1.57	(5%)	1.08	0.49	(45%)	2	106.50	1.06
MED	105.10	9.04	(8%)	64	(50%)	5.77	(63%)	5.26	0.51	(9%)	64	109.22	5.14
HIGH	101.66	8.68	(8%)	128	(100%)	9.16	(105%)	8.59	0.57	(6%)	128	105.68	8.74

Efficiency

NewJVM vs. ReuseJVM:

Reusing JVM saves time

Table 2: Experimental Results for Various Techniques

Test	All Valid			SPLat							Static Reachability (SRA)		
	NewJVM	*ReuseJVM*		*Confs*		*SPLatTime*		*IdealTime*	*Overhead*		*Confs*	*Overhead*	*Time*
101Companies (192 configs)													
LOW	35.46	2.13	(6%)	32	(16%)	1.64	(77%)	0.72	0.92	(127%)	96	84.04	1.28
MED	49.37	3.90	(7%)	160	(83%)	6.84	(175%)	3.58	3.26	(91%)	192	82.54	3.99
HIGH	283.69	45.26	(15%)	176	(91%)	47.6	(105%)	41.59	6.01	(14%)	192	81.93	45.16
Elevator (20 configs)													
LOW	10.74	5.17	(48%)	2	(10%)	1.33	(25%)	0.71	0.62	(87%)	2	23.29	0.76
MED	50.97	46.65	(91%)	10	(50%)	23.62	(50%)	23.14	0.48	(2%)	20	23.74	46.17
HIGH	62.57	59.48	(95%)	20	(100%)	60.71	(102%)	59.28	1.43	(2%)	20	24.38	60.43
Email (40 configs)													
LOW	40.63	10.74	(26%)	1	(2%)	1.00	(9%)	0.87					
MED	57.56	48.87	(84%)	30	(75%)	36.99	(75%)	37.14					
HIGH	58.02	48.93	(84%)	40	(100%)	48.96	(100%)	49.26					
GPL (73 configs)													
LOW	19.21	2.23	(11%)	6	(8%)	0.79	(35%)	0.29					
MED	190.53	171.62	(90%)	55	(75%)	130.87	(76%)	128.52					
HIGH	314.20	285.89	(90%)	70	(95%)	278.77	(97%)	277.48					
JTopas (32 configs)													
LOW	26.59	16.83	(63%)	8	(25%)	6.29	(37%)	4.49					
MED	29.04	18.55	(63%)	16	(50%)	13.16	(70%)	9.71					
HIGH	28.92	18.93	(65%)	32	(100%)	25.31	(133%)	18.43					
MinePump (64 configs)													
LOW	23.71	7.53	(31%)	9	(14%)	3.65	(48%)	1.90					
MED	59.72	14.78	(24%)	24	(37%)	10.43	(70%)	6.26					
HIGH	13.72	5.75	(41%)	48	(75%)	37.80	(657%)	4.81					
Notepad (144 configs)													
LOW	398.22	135.60	(34%)	2	(1%)	3.06	(2%)	2.45					
MED	418.23	156.27	(37%)	96	(66%)	104.95	(67%)	104.91					
HIGH	419.99	153.39	(36%)	144	(100%)	153.11	(99%)	152.16	0.94	(0%)	144	81.29	151.94
Prevayler (32 configs)													
LOW	65.34	40.23	(61%)	12	(37%)	22.49	(55%)	22.8	-0.31	(-1%)	32	205.54	45.39
MED	121.38	96.50	(79%)	24	(75%)	102.49	(106%)	105.86	-3.37	(-3%)	32	214.67	111.37
HIGH	149.08	120.7	(80%)	32	(100%)	127.17	(105%)	131.37	-4.20	(-3%)	32	290.66	135.61
Sudoku (20 configs)													
LOW	51.11	48.10	(94%)	4	(20%)	42.72	(88%)	24.12	18.6	(77%)	10	31.87	24.28
MED	118.14	105.67	(89%)	10	(50%)	58.31	(55%)	54.16	4.15	(7%)	10	31.75	53.67
HIGH	489.60	334.82	(68%)	20	(100%)	316.47	(94%)	332.36	-15.89	(-4%)	20	31.74	338.48
Xstream (128 configs)													
LOW	111.26	30.04	(27%)	2	(1%)	1.57	(5%)	1.08	0.49	(45%)	2	106.50	1.06
MED	105.10	9.04	(8%)	64	(50%)	5.77	(63%)	5.26	0.51	(9%)	64	109.22	5.14
HIGH	101.66	8.68	(8%)	128	(100%)	9.16	(105%)	8.59	0.57	(6%)	128	105.68	8.74

Efficiency

NewJVM vs. ReuseJVM:

Reusing JVM saves time

ReuseJVM vs. SPLat:

SPLat reduces time when #Confs is reduced

Table 2: Experimental Results for Various Techniques

	All Valid			SPLat							Static Reachability (SRA)		
Test	*NewJVM*	*ReuseJVM*		*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
				Elevator (20 configs)									
				2	(10%)	1.33	(25%)	0.71	0.62	(87%)	2	23.29	0.76
				10	(50%)	23.62	(50%)	23.14	0.48	(2%)	20	23.74	46.17
				20	(100%)	60.71	(102%)	59.28	1.43	(2%)	20	24.38	60.43
				Email (40 configs)									
				1	(2%)	1.00	(9%)	0.87	0.13	(14%)	1	23.62	0.87
				30	(75%)	36.99	(75%)	37.14	-0.15	(0%)	40	22.81	49.02
				40	(100%)	48.96	(100%)	49.26	-0.31	(0%)	40	23.84	49.16
				GPL (73 configs)									
				6	(8%)	0.79	(35%)	0.29	0.49	(168%)	6	104.97	0.30
				55	(75%)	130.87	(76%)	128.52	2.35	(1%)	55	99.41	128.69
				70	(95%)	278.77	(97%)	277.48	1.29	(0%)	73	103.52	286.28
				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)									
LOW	23.71	7.33	(31%)	9	(14%)	3.65	(48%)	1.90	1.75	(91%)	64	22.69	7.49
MED	59.72	14.78	(24%)	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
				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
				Prevayler (32 configs)									
				12	(37%)	22.49	(55%)	22.8	-0.31	(-1%)	32	205.54	45.39
				24	(75%)	102.49	(106%)	105.86	-3.37	(-3%)	32	214.67	111.37
				32	(100%)	127.17	(105%)	131.37	-4.20	(-3%)	32	290.66	135.61
				Sudoku (20 configs)									
				4	(20%)	42.72	(88%)	24.12	18.6	(77%)	10	31.87	24.28
				10	(50%)	58.31	(55%)	54.16	4.15	(7%)	10	31.75	53.67
				20	(100%)	316.47	(94%)	332.36	-15.89	(-4%)	20	31.74	338.48
				Xstream (128 configs)									
				2	(1%)	1.57	(5%)	1.08	0.49	(45%)	2	106.50	1.06
				64	(50%)	5.77	(63%)	5.26	0.51	(9%)	64	109.22	5.14
HIGH	101.66	8.68	(8%)	128	(100%)	9.16	(105%)	8.59	0.57	(6%)	128	105.68	8.74

SPLat vs. SRA

SPLat is more precise:

SPLat.Confs <

SRA.Confs

SPLat is more efficient:

SPLat.Overhead

< SRA.Overhead;

SPLat.IdealTime

< SRA.Time

Answer to QR1: comparison

SPLat is more efficient than ReuseJVM and SRA

SPLat prunes configurations faster and more precisely than SRA

Table 2: Experimental Results for Various Techniques

	All Valid			SPLat							Static Reachability (SRA)		
Test	*NewJVM*	*ReuseJVM*		*Confs*		*SPLatTime*		*IdealTime*	*Overhead*		*Confs*	*Overhead*	*Time*
101Companies (192 configs)													
LOW	35.46	2.13	(6%)	32	(16%)	1.64	(77%)	0.72	0.92	(127%)	96	84.04	1.28
MED	49.37	3.90	(7%)	160	(83%)	6.84	(175%)	3.58	3.26	(91%)	192	82.54	3.99
HIGH	283.69	45.26	(15%)	176	(91%)	47.6	(105%)	41.59	6.01	(14%)	192	81.93	45.16
Elevator (20 configs)													
LOW	10.74	5.17	(48%)	2	(10%)	1.33	(25%)	0.71	0.62	(87%)	2	23.29	0.76
MED	50.97	46.65	(91%)	10	(50%)	23.62	(50%)	23.14	0.48	(2%)	20	23.74	46.17
HIGH	62.57	59.48	(95%)	20	(100%)	60.71	(102%)	59.28	1.43	(2%)	20	24.38	60.43
Email (40 configs)													
LOW	40.63	10.74	(26%)	1	(2%)	1.00	(9%)	0.87	0.13	(14%)	1	23.62	0.87
MED	57.56	48.87	(84%)	30	(75%)	36.99	(75%)	37.14	-0.15	(0%)	40	22.81	49.02
HIGH	58.02	48.93	(84%)	40	(100%)	48.96	(100%)	49.26	-0.31	(0%)	40	23.84	49.16
GPL (73 configs)													
LOW	19.21	2.23	(11%)	6	(8%)	0.79	(35%)	0.29	0.49	(168%)	6	104.97	0.30
MED	190.53	171.62	(90%)	55	(75%)	130.87	(76%)	128.52	2.35	(1%)	55	99.41	128.69
HIGH	314.20	285.89	(90%)	70	(95%)	278.77	(97%)	277.48	1.29	(0%)	73	103.52	286.28
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
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
Prevayler (32 configs)													
				12	(37%)	22.49	(55%)	22.8	-0.31	(-1%)	32	205.54	45.39
				24	(75%)	102.49	(106%)	105.86	-3.37	(-3%)	32	214.67	111.37
				32	(100%)	127.17	(105%)	131.37	-4.20	(-3%)	32	290.66	135.61
Sudoku (20 configs)													
				4	(20%)	42.72	(88%)	24.12	18.6	(77%)	10	31.87	24.28
				10	(50%)	58.31	(55%)	54.16	4.15	(7%)	10	31.75	53.67
HIGH	489.60	334.82	(68%)	20	(100%)	316.47	(94%)	332.36	-15.89	(-4%)	20	31.74	338.48
Xstream (128 configs)													
LOW	111.26	30.04	(27%)	2	(1%)	1.57	(5%)	1.08	0.49	(45%)	2	106.50	1.06
MED	105.10	9.04	(8%)	64	(50%)	5.77	(63%)	5.26	0.51	(9%)	64	109.22	5.14
HIGH	101.66	8.68	(8%)	128	(100%)	9.16	(105%)	8.59	0.57	(6%)	128	105.68	8.74

Answer to QR2: Overhead

Large relative overhead for short tests (*LOW*)

Small for long tests (*HIGH*)

→ Can save total time even with large overhead

RQ3: Test on Groupon codebase

- **Groupon PWA:**
  - >171K lines of Ruby code
  - >170 (boolean) feature variables → 2^{170} configurations
  - Test code: >231K lines
  - >19K Rspec tests, each under one configuration

- **Apply SPLat:**
  - Allow varying features

    → #Configurations each test covers? (upperbound 16)

    → #Features each test encounters?

RQ3: Test on Groupon codebase

Total:
>170 features
>19K tests

Table 3: 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	-	-

"17" means >16

#Configs << 2^{170}

Table 4: 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

The maximum is 43: << 170

Answer to RQ3:

SPLat can scale to large industrial code with low implementation cost

Critiques:

- Introduced **upperbound (16) for #configs** to simplify the problem
- Assumed **no constraints** among features
- Didn't report **time cost**

Discussion

Question 1:

How do the previously mentioned critiques affect the validity of SPLat scaling to large codebases?

Discussion

Question 2:

Can SPLat scale to non-boolean variables? How?

Discussion

Question 3:

What will SPLat do if a feature is encountered but its code has no effect?

Discussion

Question 4:

Can you think of any potential optimization to improve the efficiency of SPLat?

Discussion

Question 5:

What needs to be done to apply SPLat to an SPL program?

References

1. http://www.methodsandtools.com/archive/archive.php?id=45
2. http://www.sei.cmu.edu/productlines/
3. http://resources.sei.cmu.edu/asset_files/Presentation/2008_017_001_24246.pdf

Thank you!