Faster Bug Detection for Software Product Lines with Incomplete Feature Models

Sabrina Souto
Federal University of Pernambuco
Recife, PE, Brazil

Divya Gopinath
University of Texas
Austin, TX, USA

Marcelo d'Amorim
Federal University of Pernambuco
Recife, PE, Brazil

Darko Marinov
University of Illinois
Urbana, IL, USA

Sarfraz Khurshid
University of Texas
Austin, TX, USA

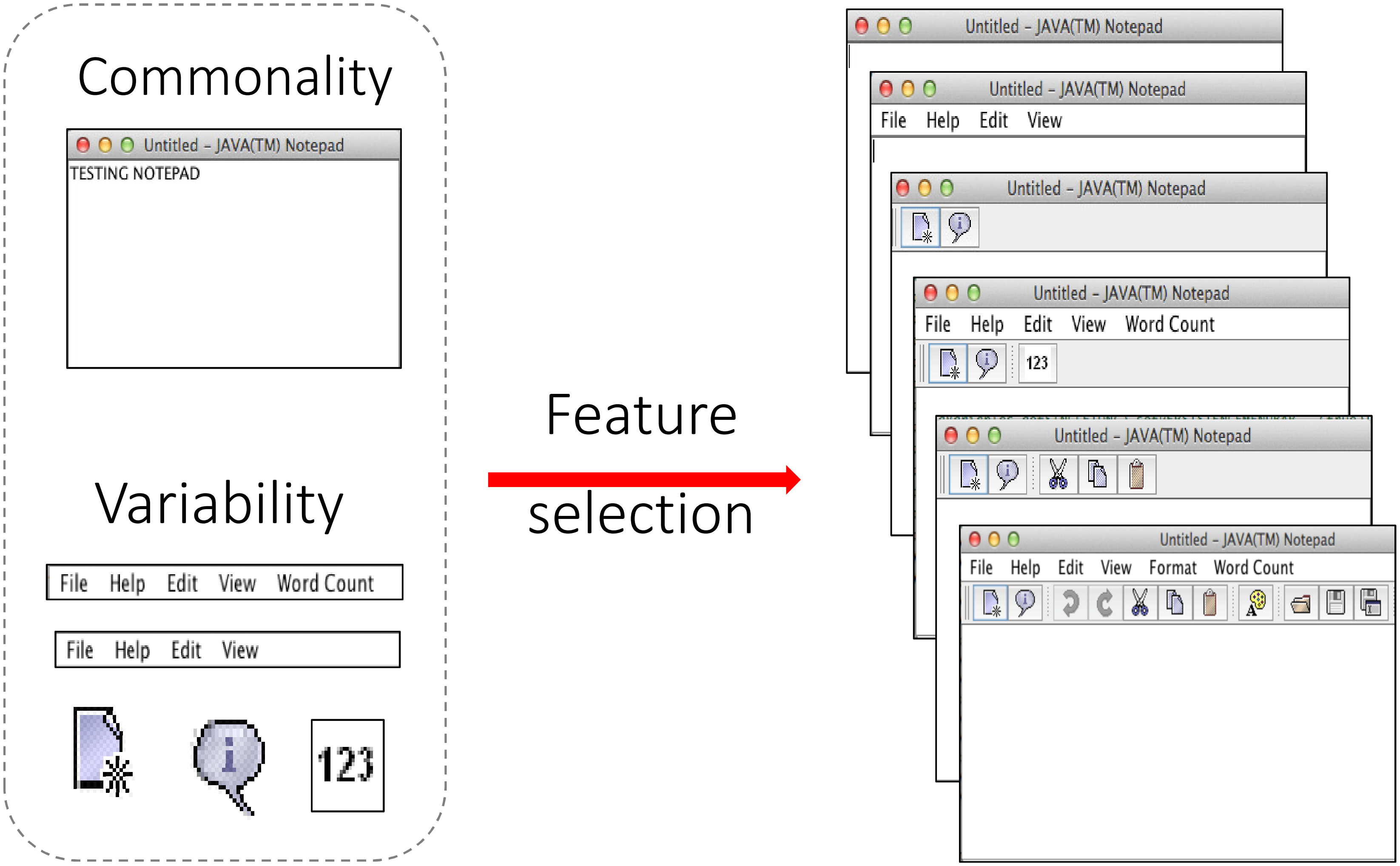
Don Batory
University of Texas
Austin, TX, USA

Ackwoledgements: US NSF CCF- 0845628, CCF-1012759, CCF-1212683, CCF-1319688, CCF- 1439957

SPLC 2015
Nashville, TN
July 23, 2015

Ackwoledgements: Brazil FACEPE BPG-0675-1.03/09 CNPq 457756/2014-4

Context: Software Product Lines



Our Research Background

- **Mostly software testing**
  - Generate new tests to find bugs
  - Run existing tests faster/better
- **Currently dominant approach**
  - Test real code (ideally from open source)
  - May use additional code artifacts (ideally real tests or comments, sometimes academic specs or more)
  - Find real bugs

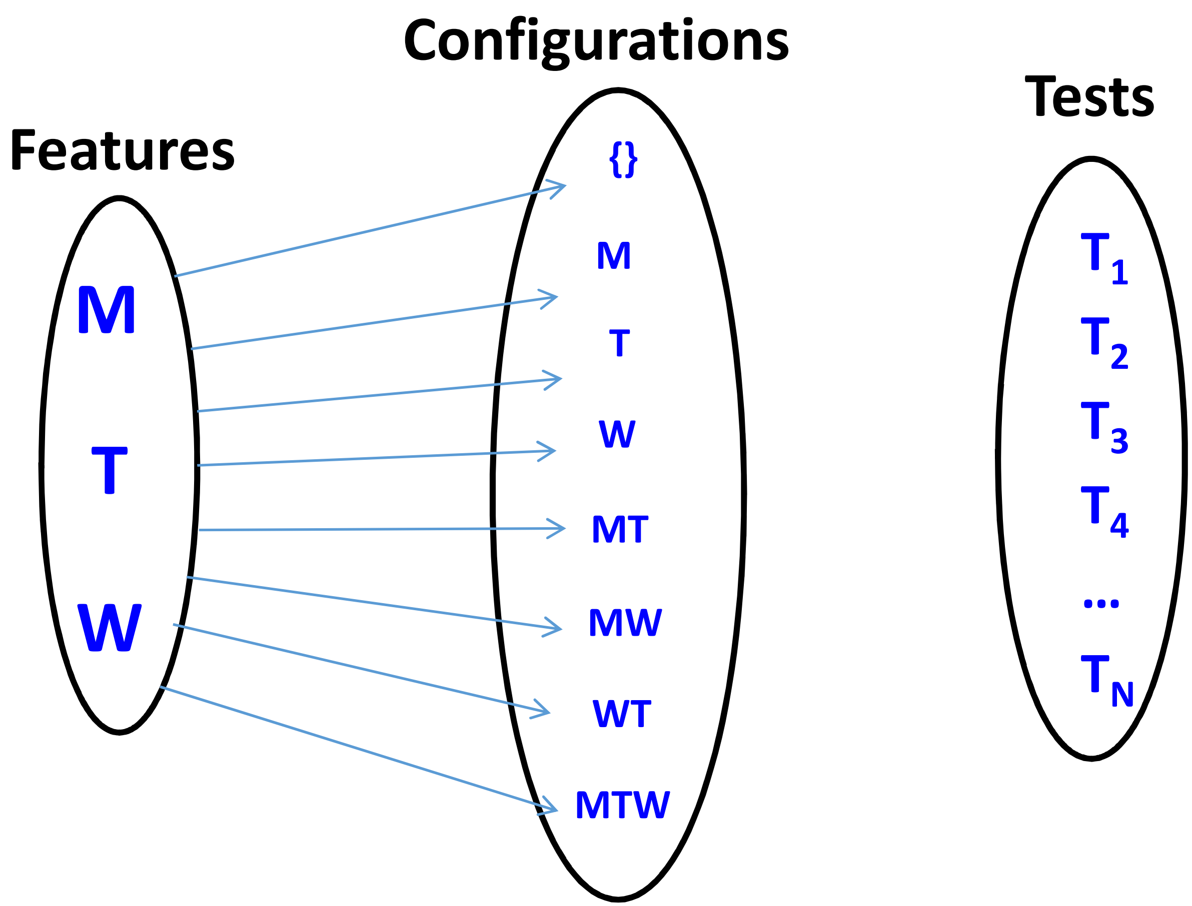
General Terminology

- **Features**
  - Functionalities of software systems
- **A Software Product Line – SPL**
  - Is a family of programs
  - Each program is defined by a unique combination of features
- **Configurations**
  - Selection of features
- **Feature Model – FM**
  - Defines a set of consistent configurations
  - *Not always documented*

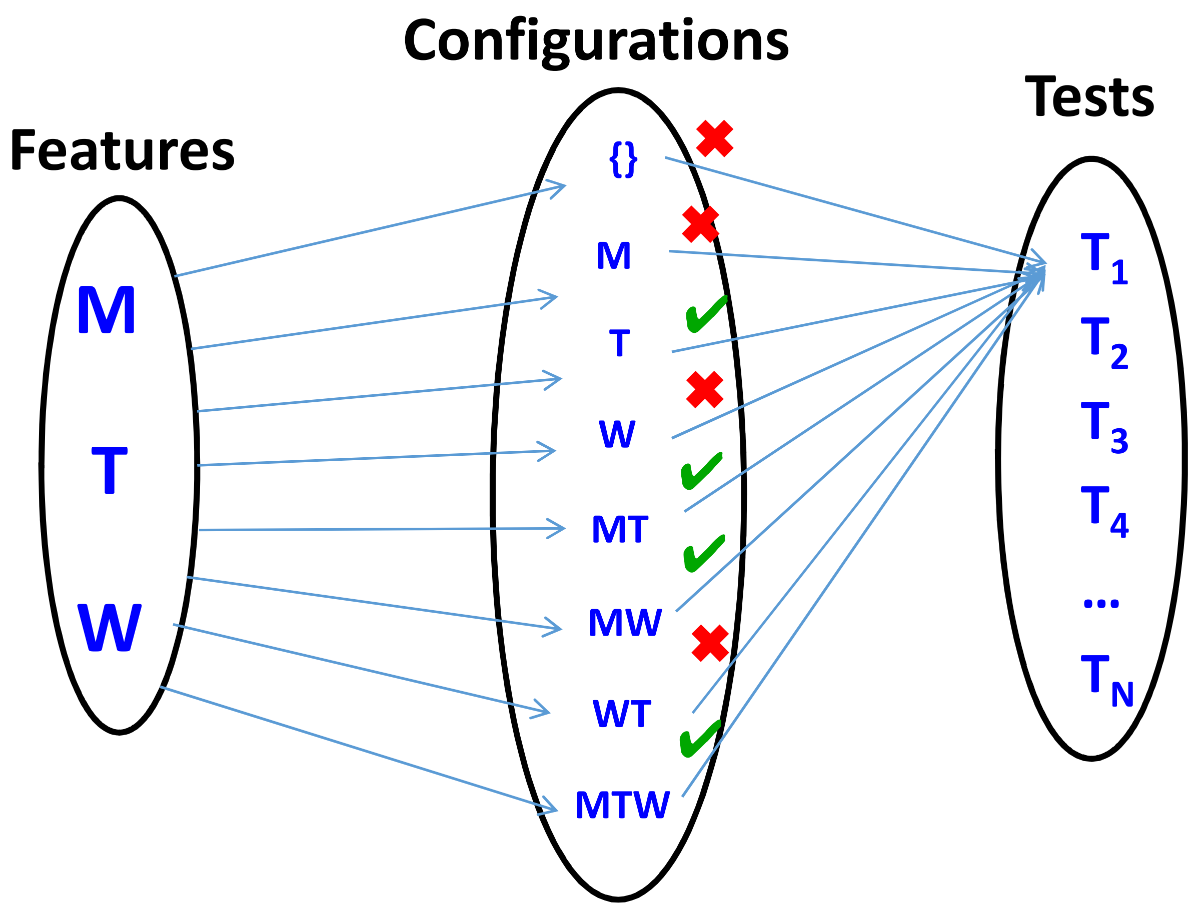
Problem:
Testing SPLs with
Incomplete Feature Model

Our Solution:
-- SPLif --

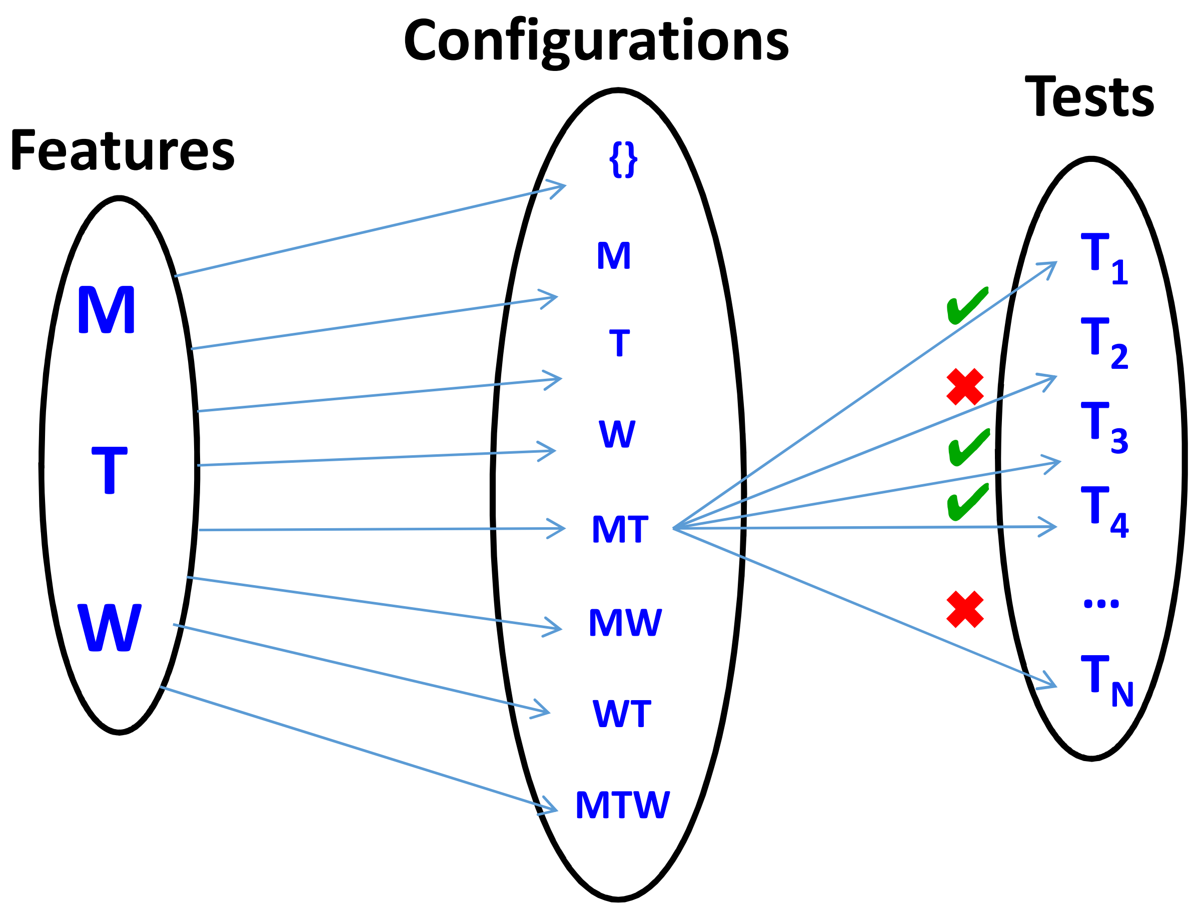
Problem
Testing SPLs with
Incomplete Feature Model



Problem
Testing SPLs with
Incomplete Feature Model

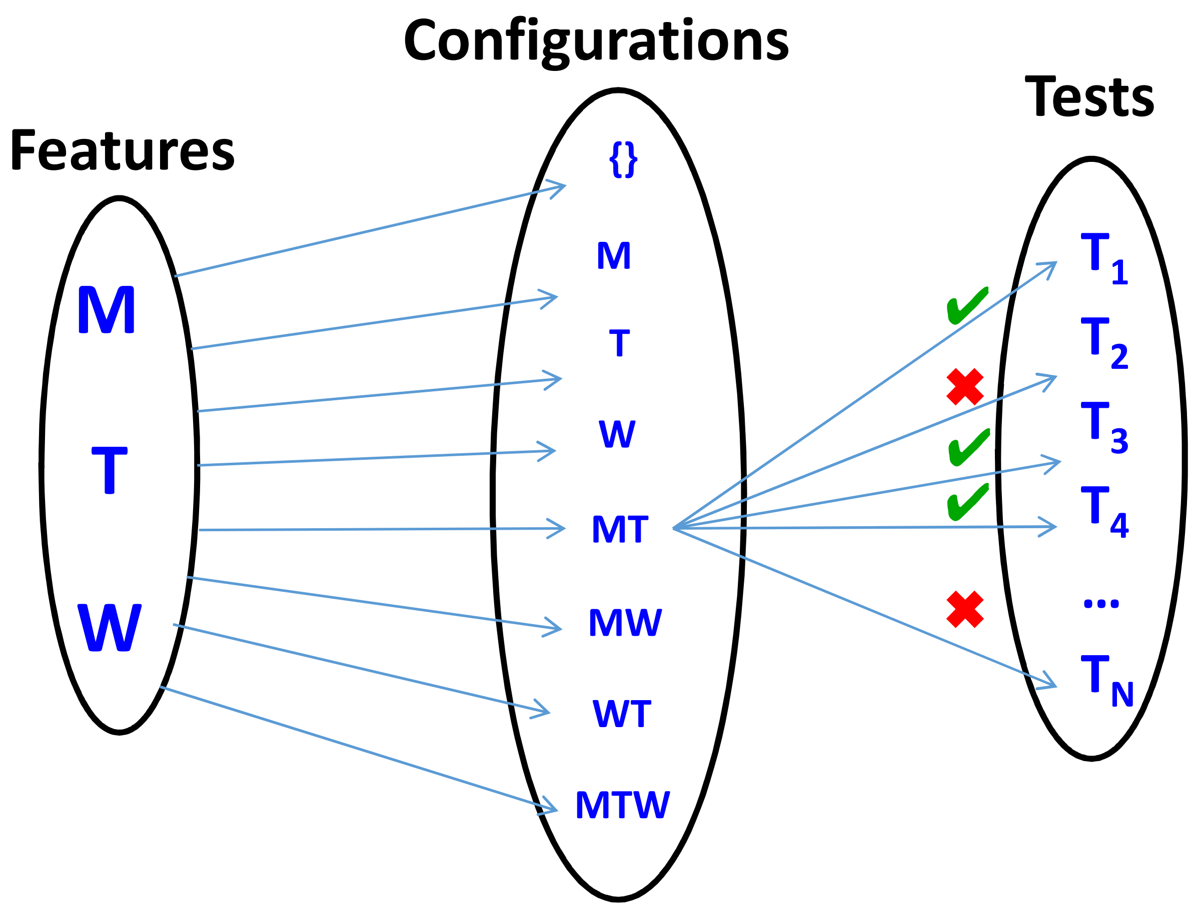


Problem
Testing SPLs with Incomplete Feature Model



Problem
Testing SPLs with Incomplete Feature Moc

The **FM** is **essential** to **distinguish** the **causes** for test **failures**, because it can detect (in)consistent configurations.

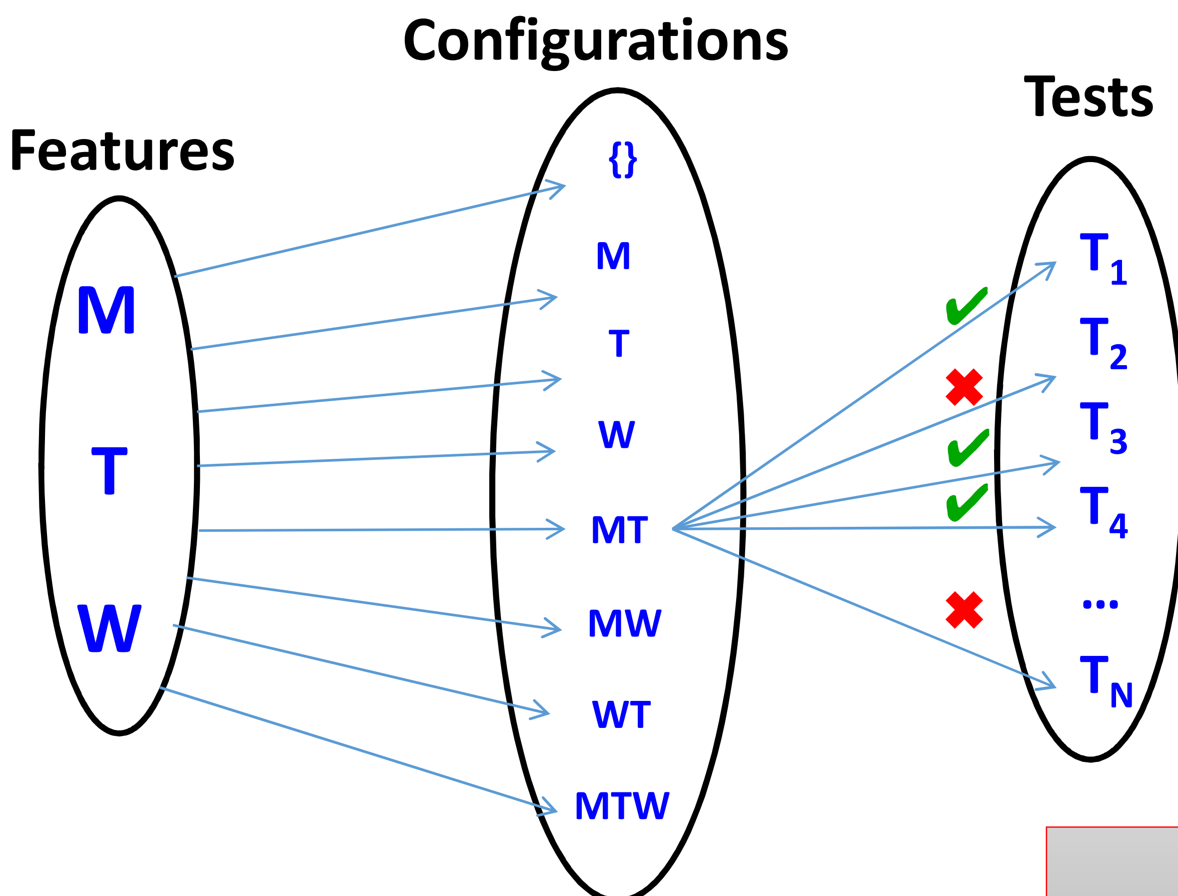


Possible causes of failures:

1. Inconsistent configurations
2. Test too restrictive
3. Bug in code

Problem
Testing SPLs with Incomplete Feature Mod

The **FM** is **essential** to **distinguish** the **causes** for test **failures**, because it can detect (in)consistent configurations.



Possible causes of failures:

1. Inconsistent configurations
2. Test too restrictive
3. Bug in code

FMs are not always available!

Problem Summary

- Feature models play a key role in testing SPLs
  - Constrain the space of configurations to test
  - Enable accurate categorization of failing tests
- Most prior work on testing SPLs assumes the availability of a complete feature model
- In practice, FMs are not always available
  - How to reduce the number of configurations per tests to run?
  - How to discover the causes for test failures?

False positives!
A test can fail due to a configuration that is not in the (absent/incomplete) model.

Related Work

- **SPL Testing**
  [Qu *et al.* ISSTA'08] [Cabral *et al.* SPLC'10] [Uzuncaova *et al.* TSE'10]
  [Garvin *et al.* ISSRE'11] [Kim *et al.* AOSD'11][Kastner *et al.* FOSD'12]
  [Kim *et al.* ISSRE'12] [Shi *et al.* FASE'12] [Song *et al.* ICSE'12]
  [Apel *et al.* ICSE'13] [Kim *et al.* FSE'13]
- **FM Extraction and Inference**
  [Czarnecki and Wasowski, SPLC'07] [Alves *et al.* SPLC'08] [Weston *et al.* SPLC'09]
  [Rabkin *et al.* ICSE'11] [She *et al.* ICSE'11] [Acher *et al.* VaMos'12]
  [Lopez-Herrejon *et al.* SSBSE'12] [Haslinger et al. FASE'13]
  [Davril *et al.* FSE'13] [Xu *et al.* SOSP'13]
- **Fault Localization**
  [Jones *et al.* ICSE'02] [Dallmeier *et al.* ECOOP'05] [Abreu *et al.* PRDC'06]
  [Abreu *et al.* TAIC'07] [Qu *et al.* ISSTA'08] [Renieris *et al.* ISSTA'08]
  [Abreu *et al.* ASE'09]
- **Configuration Troubleshooting**
  [Garvin *et al.* ASAS'12] [Zhang and Ernst *et al.* ICSE'13]
  [Zhang and Ernst *et al.* ICSE'14] [Swanson *et al.* FSE'14]

Related Work

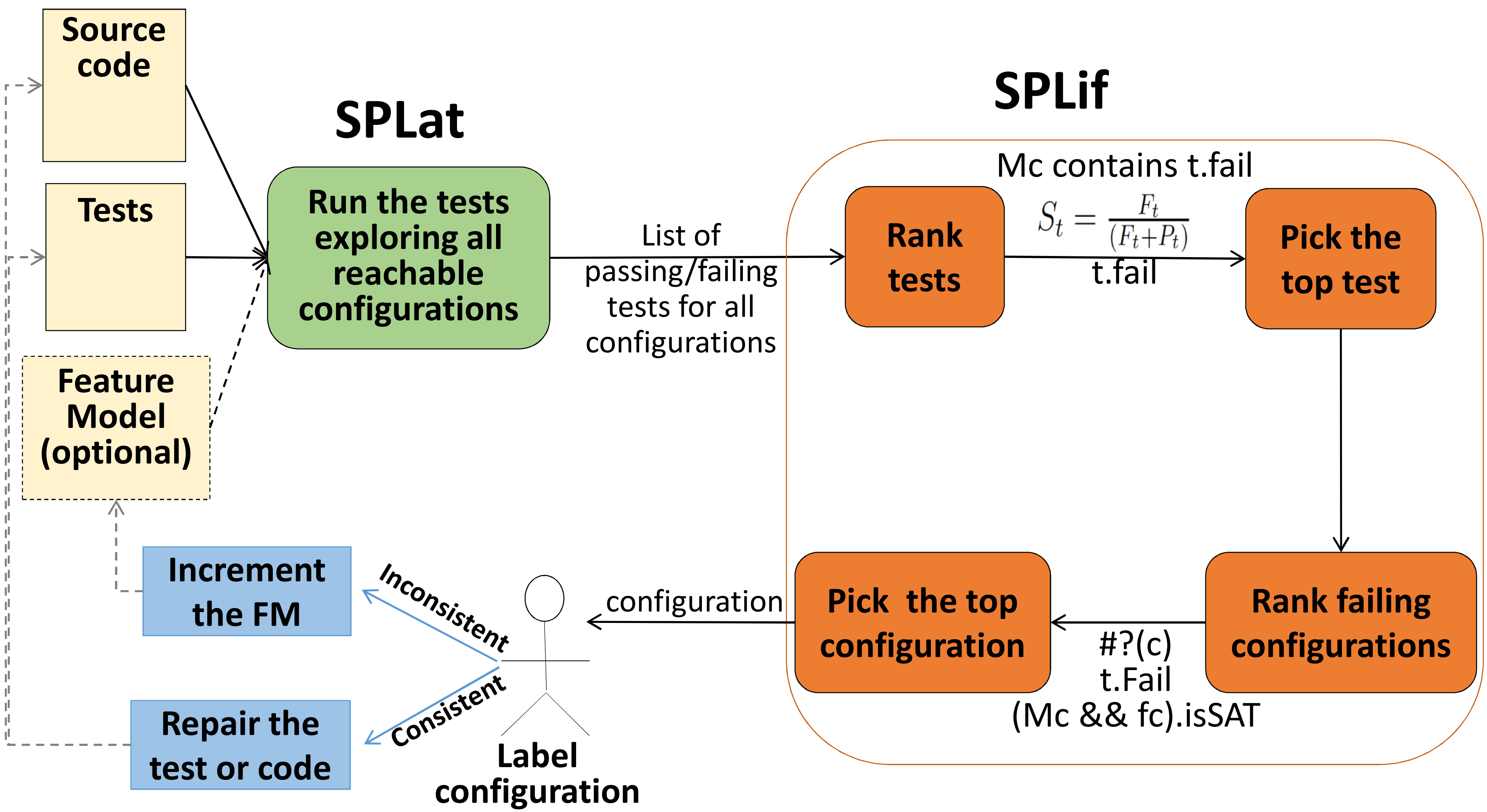
- **SPL Testing**
  [Qu *et al.* ISSTA'08] [Cabral *et al.* SPLC'10] [Uzuncaova *et al.* TSE'10]
  [Garvin *et al.* ISSRE'11] [Kim *et al.* AOSD'11][Kastner *et al.* FOSD'12]
  [Kim *et al.* ISSRE'12] [Shi *et al.* FASE'12] [Song *et al.* ICSE'12]
  [Apel *et al.* ICSE'13] [Kim *et al.* FSE'13]
- **FM Extraction and Inference**
  [Czarnecki and Wasowski, SPLC'07] [Alves *et al.* SPLC'08] [Weston *et al.* SPLC'09]
  [Rabkin *et al.* ICSE'11] [She *et al.* ICSE'11] [Acher *et al.* VaMos'12]
  [Lopez-Herrejon *et al.* SSBSE'12] [Haslinger et al. FASE'13]
  [Davril *et al.* FSE'13] [Xu *et al.* SOSP'13]
- **Fault Localization**
  [Jones *et al.* ICSE'02] [Dallmeier *et al.* ECOOP'05] [Abreu *et al.* PRDC'06]
  [Abreu *et al.* TAIC'07] [Qu *et al.* ISSTA'08] [Renieris *et al.* ISSTA'08]
  [Abreu *et al.* ASE'09]
- **Configuration Troubleshooting**
  [Garvin *et al.* ASAS'12] [Zhang and Ernst *et al.* ICSE'13]
  [Zhang and Ernst *et al.* ICSE'14] [Swanson *et al.* FSE'14]

No prior work combines FM inference with tests and their executions

Insight

- Tests that fail on consistent configurations indicate real faults
- We need to find fault-revealing consistent configurations soon
  - Enable efficient bug detection
- The FM is not available or is incomplete
  - Do not need to discover the entire FM
  - Discover only the relevant part to check the consistency of the fault-revealing configuration
- Assumption
  - The developer/user will help to check such consistency
  - The developer/user is aware about many feature relationships

Proposal: SPLif



Specific Terminology

- Each feature can assume 3 values:
  - **0**: the feature is **disabled** (=false)
  - **1**: the feature is **enabled** (=true)
  - **?**: the feature has no value yet (=**unknown**)
- **Incomplete** vs. **Complete** Configuration

MTW=0?1 (incomplete)
MTW=010 (complete)

Notepad Features:
Menubar, **T**oolbar, and **W**ordcount

- **Consistent** vs. **Inconsistent** Configuration

MTW=0?1 (consistent)
MTW=00? (inconsistent)

Notepad Constraint: $\mathbf{M} \vee \mathbf{T}$
(Initially Undocumented)

SPLif on Notepad (1 test)

- Configurations (MTW):

  111

  011

  110

  010

  10?

  00?

```
class Notepad {
  void toolBar() {
    if(T) {
      ...
      if(W)
        ...
    }

    if (M) { ... }
  }

  ...

  void test() {
    toolBar();
  }
}
```

SPLif Notepad (1 test)

- Configurations (MTW):

111

011 ✖

110

010

10? ✖

00? ✖

Execution of some tests fails!

SPLif Notepad (1 test)

- Configurations (MTW):

Select failing configurations

011 ✖

10? ✖

00? ✖

SPLif Notepad (1 test)

- Configurations (MTW):

00**?**

10**?**

011

Rank configurations for inspection

SPLif Notepad (1 test)

- Configurations (MTW):

00? Inconsistent!

10?

011

SPLif Notepad (1 test)

- Configurations (MTW):

00?

10?

011

$\boxed{\text{Inconsistent!}}$

Partial Feature Model (PFM) = $!(\bigcup c_i)$, where c_i is an inconsistent configuration

In this case c_i=(`!M ∧ !T`) and PFM=

`!(!M ∧ !T)`

`!!M ∨ !!T`

`M ∨ T`

SPLif Notepad (1 test)

- Configurations (MTW):

00?

Inconsistent!

10?

011

Partial Feature Model (PFM) = !($\cup$ c_i),
where c_i is an inconsistent configuration

In th

`! (!M ∧ !`

`! !M ∨ !`

`M ∨ T`

Configurations that violate this constraint will not be inspected!

SPLif Notepad (1 test)

- Configurations (MTW):

00?

10?

011

Consistent

The test failed on configurations where no inconsistency has been observed. User should inspect test and/or code!

Partial Feature Model:

$\mathbf{M} \vee \mathbf{T}$

Evaluation: Setup

- **Questions**
  - **RQ1:** How well does SPLif rank faulty tests for inspection?
  - **RQ2:** How well does SPLif rank configurations (of selected tests) for inspection?
- **Experiment**
  - 5 SPLs previously used
  - The tests used were created by students
  - 4 techniques:
    - Random
    - Memory
    - Weighted
    - Adaptive

Evaluation: Results

Ranking Tests

RQ1: How well does SPLif rank faulty tests for inspection?

Companies

R	t_i	S_i
1	15	0.75
2	16	0.75
3	19	0.75
4	14	0.75
5	18	0.58
6	13	0.50
7	17	0.50

GPL

R	t_i	S_i
1	24	0.97
2	22	0.88
3	21	0.75
4	18	0.50
5	19	0.50
6	25	0.50
7	23	0.50
8	20	0.30

Notepad

R	t_i	S_i
1	31	0.59
2	29	0.52
3	30	0.42

DesktopSearcher

R	t_i	S_i
1	24	1.00
2	25	1.00
3	39	1.00
4	35	1.00
5	36	1.00
6	42	0.94
7	40	0.93
8	43	0.90
9	41	0.88
10	37	0.88
11	38	0.88
12	32	0.75
13	33	0.75
14	34	0.75
15	29	0.60
16	13	0.33
17	27	0.27
18	28	0.27
19	30	0.27
20	10	0.20
21	26	0.20
22	21	0.20
23	11	0.11
24	12	0.11
25	14	0.11
26	15	0.11
27	16	0.11
28	17	0.11
29	19	0.11
30	22	0.11
31	31	0.04
32	18	0.03
33	20	0.03
34	23	0.03

ZipMe

R	t_i	S_i
1	59	0.75
2	62	0.51
3	30	0.50
4	31	0.50
5	60	0.50
6	42	0.50
7	53	0.50
8	48	0.50
9	61	0.44
10	50	0.42
11	58	0.33
12	35	0.33
13	41	0.33
14	32	0.25
15	33	0.25
16	36	0.25
17	37	0.25
18	38	0.25
19	39	0.25
20	40	0.25
21	47	0.25
22	43	0.25
23	44	0.25
24	51	0.21
25	52	0.20
26	46	0.20
27	45	0.14
28	55	0.14
29	54	0.13
30	56	0.13
31	34	0.11
32	49	0.11
33	57	0.05

Evaluation: Results

Ranking Configurations

RQ2: How well does SPLif rank configurations (of selected tests) for inspection?

Total Number of Inspections for All Modes					
Mode	**Companies**	**GPL**	**Notepad**	**Desktop Searcher**	**ZipMe**
Random	146	257	90	44	269
UpdateFM	69	211	40	30	45
Weighted and Adaptive	69	223	10	34	49

Case Study: GCC

- **RQ3**: How well does SPLif scale to real code?
- **Experiment**
  - Applied SPLif against the GNU Compiler Collection
    - 27 years of work from 500+ contributors
    - 7+ Million LOCs
    - 17K+ tests
    - More than 2k configuration variables (not only boolean)

GCC Evaluation: Setup

- Tests
  - 4,108 tests from 3 suites (gcc-dg, dg-torture, tree-ssa)
  - 50 configurations per test
  - Randomized SPLat execution to sample different (reachable) configurations
- Options
  - 40 most frequently cited options in the GCC bug reports
  - Initial model (incomplete) built on the work of [Garvin *et al.* ASAS'13]
- Failures
  - Inspection of failures on crashes

GCC Evaluation: Results

- Recall
  - We focused only on crash failures
  - We ran each test against 50 reachable configurations
- 4,108 tests analyzed
  - 497 tests failed (due to crash or not)
  - 3,986 pairs of tests and configurations failed (due to crash or not)
- Considering only crashes
  - 43 tests manifested crashes in 268 pairs of test and configurations

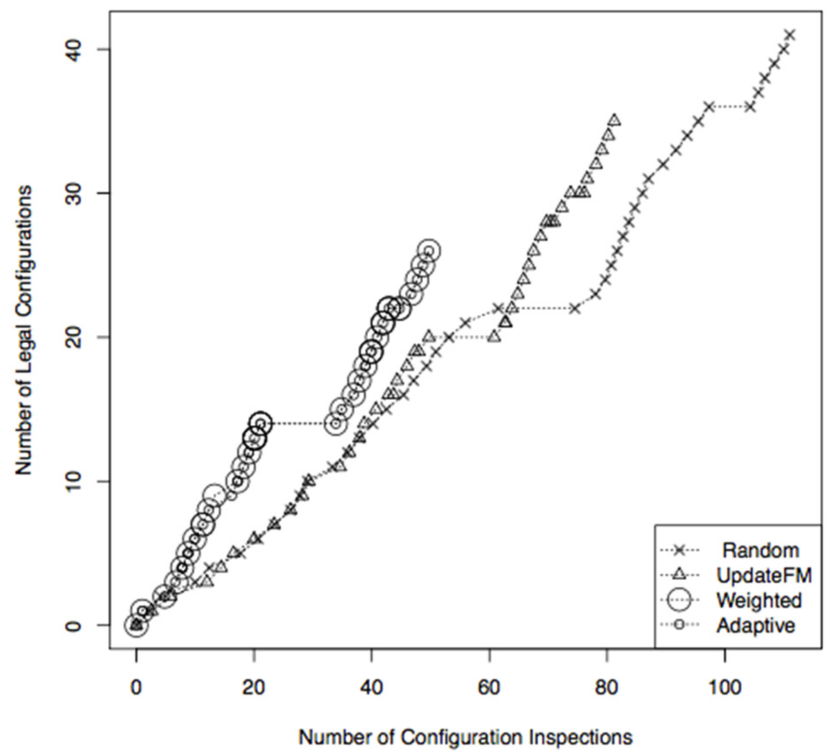
GCC Evaluation: Results

RQ3: How well does SPLif scale to real code?

Ranking Tests

GCC		
R	t_i	S
1	4069	1,00
2	4070	0,54
3	4064	0,51
4	4068	0,50
5	4066	0,46
6	4067	0,44
7	4062	0,43
8	4065	0,42
9	4063	0,40
10	4060	0,36
11	4061	0,36
12	4059	0,34
13	4055	0,32
14	4056	0,32
15	4057	0,32
16	4058	0,32
17	4044	0,31
18	4036	0,29
19	4053	0,29
20	4051	0,28
21	4052	0,28
22	4054	0,28
23	4029	0,27
24	4039	0,26
25	4045	0,26
26	4049	0,26
27	4050	0,26
28	4046	0,24
29	4047	0,24
30	4048	0,24
31	4043	0,23
32	4040	0,20
33	4041	0,20
34	4042	0,20
35	4038	0,19
36	4037	0,18
37	4034	0,18
38	4028	0,14
39	4035	0,14
40	4033	0,12
41	4032	0,12
42	4031	0,04
43	4030	0,02

Ranking Configurations



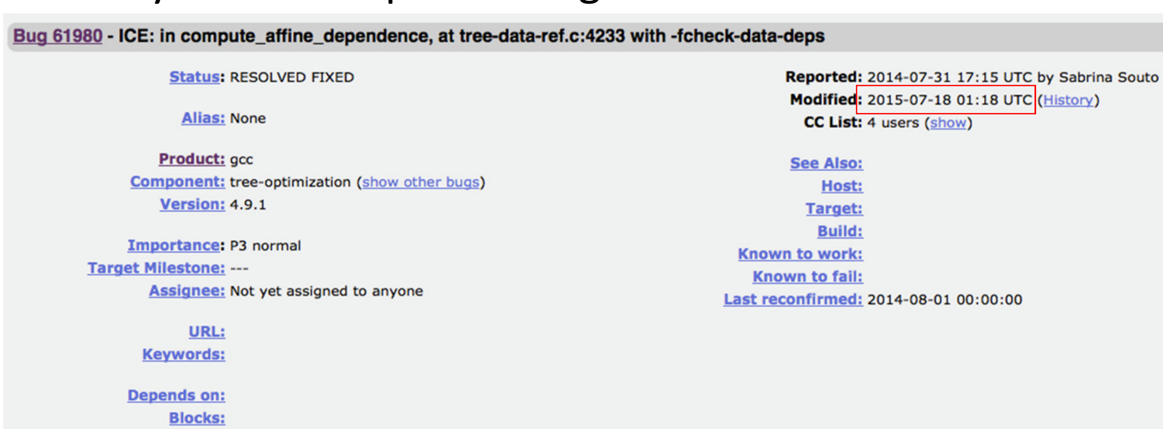
GCC Evaluation: Results

RQ3: How well does SPLif scale to real code?

New bugs found

Cluster data			Bug report data			
Name	*#Tests*	*#Pairs*	*Id*	*Confirmed*	*Fixed*	*Status*
compute_affine_dependence, tree-data-ref.c: 4233	34	223	61980	Aug.1,2014	-	NEW
int_cst_value, tree.c: 10625	4	34	62069	Aug.8,2014	-	NEW
verify_ssa failed, tree-ssa.c: 1056	1	6	62070	Aug.8,2014	Aug.11,2014	RESOLVED FIXED
build2_stat, tree.c: 4265	1	4	62140	Aug.14,2014	Oc.16,2014	RESOLVED FIXED
Segmentation fault: 11	1	1	62141	Aug.14,2014	Nov.19,2014	RESOLVED FIXED

Recently the first reported bug has been also fixed



Bug 61980 - ICE: in compute_affine_dependence, at tree-data-ref.c:4233 with -fcheck-data-deps

Status: RESOLVED FIXED
Alias: None
Product: gcc
Component: tree-optimization (show other bugs)
Version: 4.9.1
Importance: P3 normal
Target Milestone: ---
Assignee: Not yet assigned to anyone
URL:
Keywords:
Depends on:
Blocks:

Reported: 2014-07-31 17:15 UTC by Sabrina Souto
Modified: 2015-07-18 01:18 UTC (History)
CC List: 4 users (show)
See Also:
Host:
Target:
Build:
Known to work:
Known to fail:
Last reconfirmed: 2014-08-01 00:00:00

Conclusions

- The FM can detect (in)consistent configurations
  - It is essential to distinguish the causes for test failures
- Prior research assumes that SPLs come equipped with complete, formally specified FMs
  - This assumption does not always hold in practice
- We proposed SPLif
  - A new approach for testing SPLs with incomplete/absent FM
- Experiments show that SPLif
  - Helps the user prioritize failing tests and configurations for inspection
  - Is promising and can scale to large systems, such as GCC