UNIVERSITEIT STELLENBOSCH UNIVERSITY

UNIVERSITY OF Southampton

Fast Model-Based Fault Localisation with Test Suites

Geoff Birch[1], Bernd Fischer[2], and Michael R. Poppleton[1]

funded by

EPSRC

Engineering and Physical Sciences Research Council

[1] University of Southampton; [2] Stellenbosch University

Motivation

Model-based fault localisation: Ideal for code in languages with good support for **formal specifications**, developed with significant effort put into encoding that specification into the code.

Agile and **Test-Driven Development** requires rapid iteration, fluid specifications, and often provides **test suites** to express that spec.

Most work at localising faults on these **test suite specified** programs in languages like C focus on **spectrum-based** methods. This is very fast.

What if **model-based fault localisation** used **test suites** and was also fast?

Overview of Tool Stages

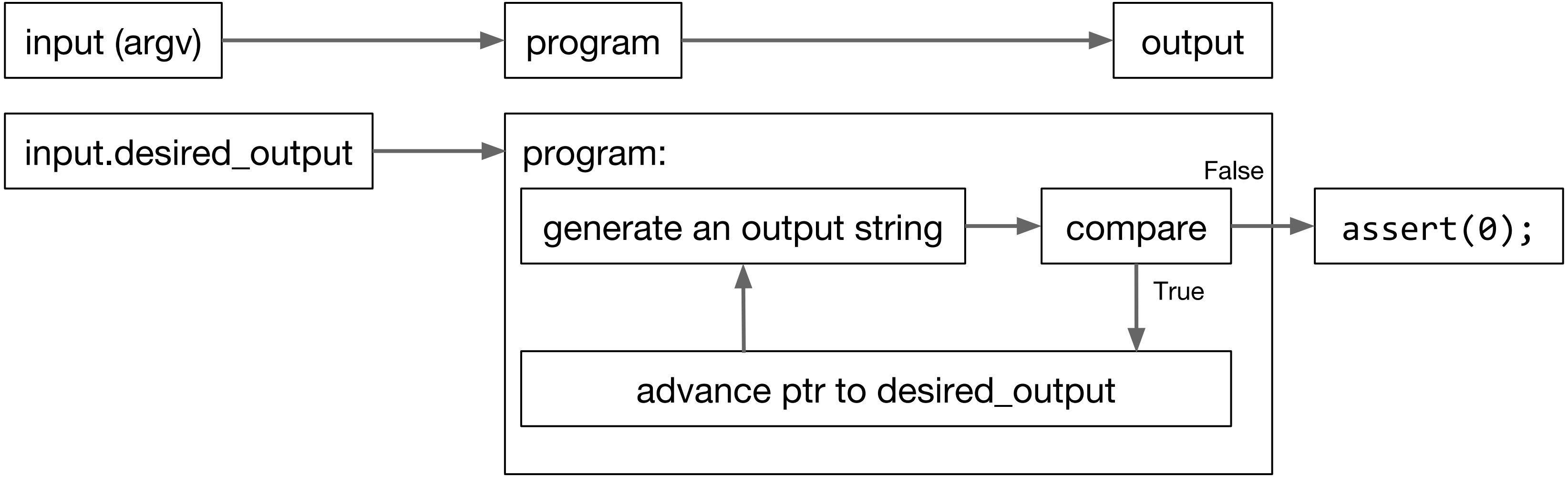
1. Transform
    a. Convert **Input Program** to **Model**;
    b. Invert **Model**;
    c. Add **Non-Determinism**.

2. Search a **failing test case** - Collect **code locations** where a change may exist that makes this into a **passing test case**.

3. Make it **fast** to search the entire test suite
    a. Use gathered search results to **minimise search space**.
    b. Complete each **test case search** as a task in a **worker pool**.

1. Transform - a) Convert

```
printf("%d", val); /*  ->  */ assert(val == atoi(argv[X]));
```




```
assert('\0' == *__end_ptr); /* or */
assert(strlen(__out_string) == __end_ptr - __out_string);
```

1. Transform - b) Invert

Counter-example raising **assert**s **become** program flow narrowing **assume**s:

```
assert(expr); /*  ->  */ klee_assume(expr);
```

New **assert**s guarantee reaching the end of the original program is now going to raise a spec fail and so **create a counter-example**:

```
exit(val); /*  ->  */ assert(0); exit(val);
```

```
main() { ... return val;} /*  ->  */ main() { ... assert(0); return val;}
```

```
main() { ... } /*  ->  */ main() { ... assert(0);}
```

1. Transform - c) Non-Determinism

Create a **toggle** and allow it to **take any value** in our range of locations:

`uint __toggle; klee_make_symbolic(&__toggle); klee_assume(__toggle < `**max**`);`

Convert all assignments to check toggle, **assign any value when active**:

`var = a + b;` `/*  ->  */ var = ((__toggle == `**val**`)? klee_sym_int() : a + b);`

Create a toggle and allow it to take any value in our range of locations:

`klee_assume(__toggle != `**val**`);`

2. Search - the Slow Way

	loc_1	loc_2	loc_3	loc_4	loc_5	loc_6	loc_7	loc_8	loc_9	loc_10
ftc_01	√	√	√			√		√	√	
ftc_02		√		√	√	√	√	√	√	
ftc_03	√	√		√		√	√	√		√
ftc_04		√	√		√	√	√	√		√
ftc_05	√	√		√		√	√	√	√	
ftc_06	√	√	√			√				
ftc_07		√		√	√	√		√	√	√
...		√				√				

2. Fast Search in a Failing Test Case

I. Encode narrowing from **locations still being searched**.

II. Call **symbolic analyser** on **non-determinism inserted, inverted model**.

III. Collect all **counter-examples** flagging **unconditional assertion failure**.

IV. Extract the `__toggle` values that indicate the **code locations** of repairs.

V. Return this **set of locations** to the **manager process**.

2. Search - the Fast Way

	loc_1	loc_2	loc_3	loc_4	loc_5	loc_6	loc_7	loc_8	loc_9	loc_10
ftc_01	√	√	√			√		√	√	
ftc_02		√		√	√	√	√	√	√	
ftc_03	√	√		√		√	√	√		√
ftc_04		√	√		√	√	√	√		√
ftc_05	√	√		√		√	√	√	√	
ftc_06	√	√	√			√				
ftc_07		√		√	√	√		√	√	√
...		√				√				

2. Fast Search in a Failing Test Case

I. Encode narrowing from **locations still being searched**.

II. Call **symbolic analyser** on **non-determinism inserted, inverted model**.

III. Collect all **counter-examples** flagging **unconditional assertion failure**.

IV. Extract the `__toggle` values that indicate the **code locations** of repairs.

V. Return this **set of locations** to the **manager process.**

3. Worker Pool - the Full Search

Start by queueing first failing test cases as **search tasks** onto **workers**.

While **Workers** have **Tasks**:

    If Has **Completed Tasks** then:

        Sleep for 25% of **Fastest Task Complete Time**.

        Locations Still Being Searched = Locations Returned by All Tasks.

        If Repeated **No Improvement** in Locations then RETURN.

        Enqueue any Active Test Cases **Not Completed**. (end of queue)

        Remove All Tasks from Workers.

        If Failing Test Cases Remain then Queue Search Tasks onto Workers.

RETURN.

The Competition

Griesmayer et al. (G) *Automated Fault Localization for C Programs*. (2007) uses CBMC on a 2.8GHz Pentium 4. [§4, Table 1, p. 104]

Jose & Majumdar (J) *Cause Clue Clauses: Error Localization Using Maximum Satisfiability*. (2011) uses MSUnCORE on a 3.16GHz Core2Duo [§6, T1, p. 443]

Our **naive (N)** reimplementation uses ESBMC v1.17 on a 3GHz Core2Duo E8400; our new algorithm using **ESBMC (E)** and **KLEE (K)** as back-end both ran on a 3.1GHz Core i5-2400, with the ESBMC v1.21 and KLEE (for LLVM 3.4) symbolic analysers.

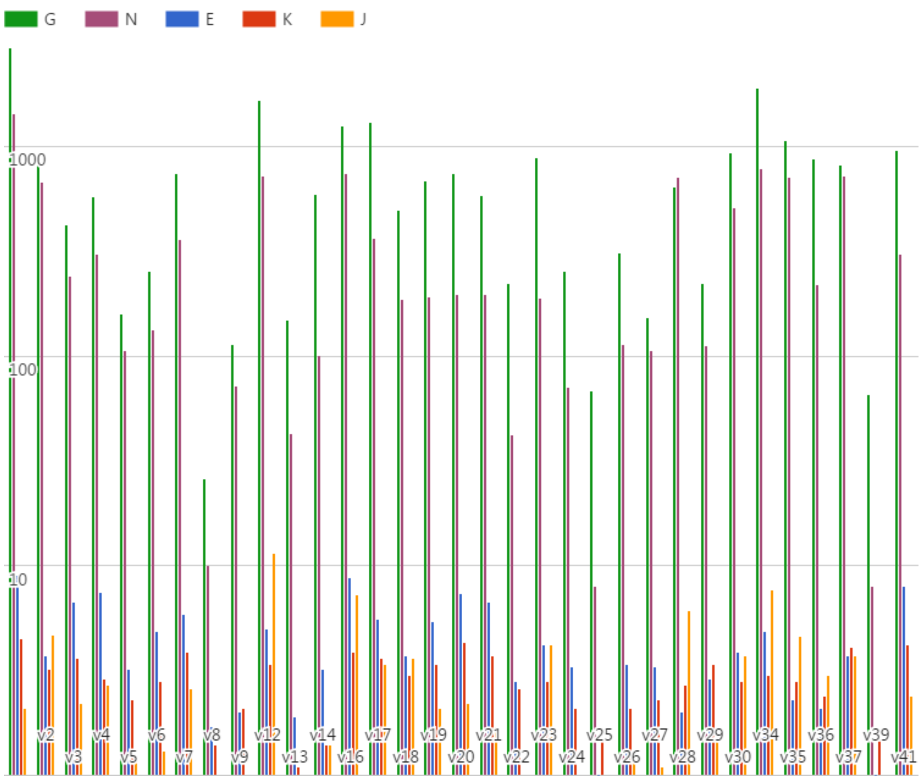
State of the art spectrum-based fault localisation methods have recently been compared using theoretical frameworks. We take results from **Naish et al.** *A Model for Spectra-based Software Diagnosis*. (2011)

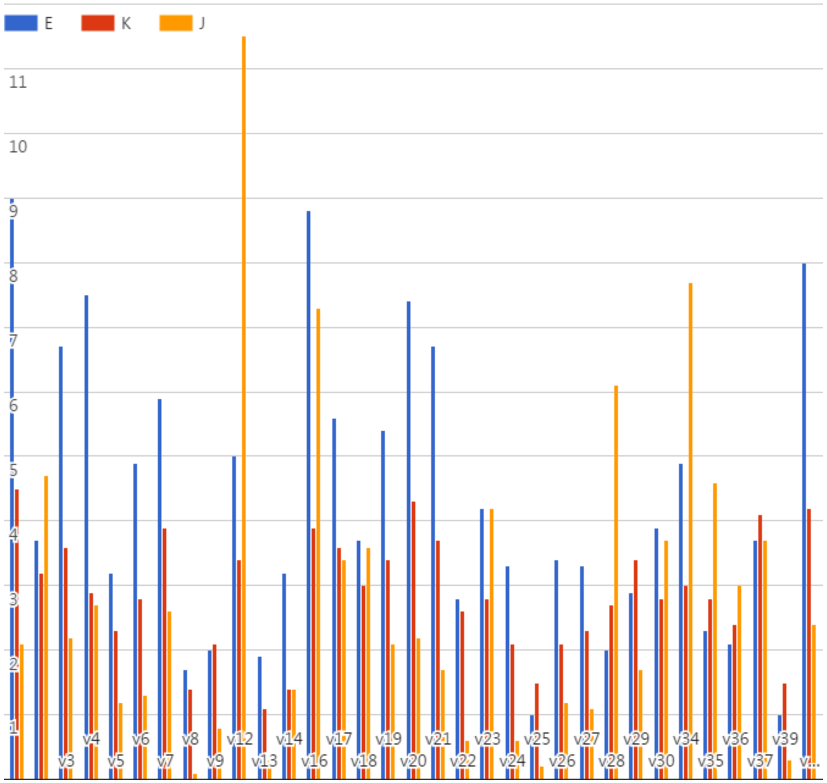
Run Time Performance

	G	N	E	K	J
v1	2953	1442	9.0	4.5	**2.1**
v2	836	678	3.7	**3.2**	4.7
v3	423	240	6.7	3.6	**2.2**
v4	576	307	7.5	2.9	**2.7**
v5	159	106	3.2	2.3	**1.2**
v6	253	134	4.9	2.8	**1.3**
v7	743	359	5.9	3.9	**2.6**
v8	26	10	1.7	1.4	**0.1**
v9	114	72	2.0	2.1	**0.8**
v12	1664	727	5.0	**3.4**	11.5
v13	149	43	1.9	1.1	**0.3**
v14	594	101	3.2	**1.4**	**1.4**
v16	1263	746	8.8	**3.9**	7.3
v17	1300	365	5.6	3.6	**3.4**
v18	499	188	3.7	**3.0**	3.6
v19	691	193	5.4	3.4	**2.1**
v20	748	196	7.4	4.3	**2.2**
v21	585	197	6.7	3.7	**1.7**
v22	223	42	2.8	2.6	**0.6**
v23	885	189	4.2	**2.8**	4.2
v24	254	71	3.3	2.1	**0.6**
v25	68	8	1.0	1.5	**0.2**
v26	311	114	3.4	2.1	**1.2**
v27	153	107	3.3	2.3	**1.1**
v28	642	711	**2.0**	2.7	6.1
v29	224	112	2.9	3.4	**1.7**
v30	939	508	3.9	**2.8**	3.7
v34	1906	790	4.9	**3.0**	7.7
v35	1069	711	**2.3**	2.8	4.6
v36	877	219	**2.1**	2.4	3.0
v37	822	729	**3.7**	4.1	**3.7**
v39	66	8	1.0	1.5	**0.3**
v41	956	309	8.0	4.2	**2.4**

Table 1. Seconds to Return Location Set for Test Suite. Griesmayer's original data [12, Table 1, p. 104] (**G**). Naïve reimplementation of Griesmayer's algorithm (**N**). New algorithm using ESBMC (**E**) and KLEE (**K**) as back-end. Jose and Majumdar's results [18, Table 1, p. 443] (**J**).

Run Time Performance





Run Time Performance

	G	N	E	K	J
v1	2953	1442	9.0	4.5	**2.1**
v2	836	678	3.7	**3.2**	4.7
v3	423	240	6.7	3.6	**2.2**
v4	576	307	7.5	2.9	**2.7**
v5	159	106	3.2	2.3	**1.2**
v6	253	134	4.9	2.8	**1.3**
v7	743	359	5.9	3.9	**2.6**
v8	26	10	1.7	1.4	**0.1**
v9	114	72	2.0	2.1	**0.8**
v12	1664	727	5.0	**3.4**	11.5
v13	149	43	1.9	1.1	**0.3**
v14	594	101	3.2	**1.4**	**1.4**
v16	1263	746	8.8	**3.9**	7.3
v17	1300	365	5.6	3.6	**3.4**
v18	499	188	3.7	**3.0**	3.6
v19	691	193	5.4	3.4	**2.1**
v20	748	196	7.4	4.3	**2.2**
v21	585	197	6.7	3.7	**1.7**
v22	223	42	2.8	2.6	**0.6**
v23	885	189	4.2	**2.8**	4.2
v24	254	71	3.3	2.1	**0.6**
v25	68	8	1.0	1.5	**0.2**
v26	311	114	3.4	2.1	**1.2**
v27	153	107	3.3	2.3	**1.1**
v28	642	711	**2.0**	2.7	6.1
v29	224	112	2.9	3.4	**1.7**
v30	939	508	3.9	**2.8**	3.7
v34	1906	790	4.9	**3.0**	7.7
v35	1069	711	**2.3**	2.8	4.6
v36	877	219	**2.1**	2.4	3.0
v37	822	729	**3.7**	4.1	**3.7**
v39	66	8	1.0	1.5	**0.3**
v41	956	309	8.0	4.2	**2.4**

Table 1. Seconds to Return Location Set for Test Suite. Griesmayer's original data [12, Table 1, p. 104] (**G**). Naïve reimplementation of Griesmayer's algorithm (**N**). New algorithm using ESBMC (**E**) and KLEE (**K**) as back-end. Jose and Majumdar's results [18, Table 1, p. 443] (**J**).

Localisation Conceptualisation

	loc_1	loc_2	loc_3	loc_4	loc_5	loc_6	loc_7	loc_8	loc_9	loc_10
ftc_01	8	15	-45			4096		22	0	
ftc_02		15		√	√	4069	√	18	7	
ftc_03	√	15		√		0	√	22		√
...		...				...				

```
int loc_2[] = {15, 15, 15, ...};
int loc_6[] = {4096, 4069, 0, ...};
```

Pass in **failing test case number**, assign to `ftc_no`:

```
var = a + b; /* @loc_2 ->  */ var = loc_2[ftc_no];
```

Localisation Performance

	G	N	K	J
v1	8.7	10.4	**7.5**	8.6
v2	**2.9**	**2.9**	**2.9**	4.6
v3	**4.0**	8.7	5.2	9.8
v4	8.7	11.0	**8.1**	9.2
v5	4.0	8.1	**3.5**	8.6
v6	7.5	11.0	**6.9**	8.6
v7	2.3	**1.7**	2.3	9.2
v8	11.0	16.8	10.4	**8.6**
v9	5.2	**4.6**	**4.6**	5.2
v12	**4.0**	4.6	**4.0**	9.2
v13	**5.2**	9.2	**5.2**	9.2
v14	**2.3**	2.9	2.9	8.1
v16	8.7	10.4	**7.5**	9.2
v17	2.3	**1.7**	2.3	9.2
v18	**2.3**	**2.3**	**2.3**	6.9
v19	2.3	**1.7**	2.3	9.2
v20	8.7	12.1	**8.1**	9.2
v21	8.7	12.7	**8.1**	8.6
v22	4.6	**4.0**	4.6	5.7
v23	5.2	**4.6**	**4.6**	6.3
v24	8.7	11.0	**7.5**	8.6
v25	4.6	**3.5**	4.0	6.9
v26	4.6	6.9	**4.0**	9.2
v27	4.0	8.1	**3.5**	10.9
v28	**1.2**	**1.2**	**1.2**	5.7
v29	**1.7**	2.3	2.3	5.7
v30	**2.3**	2.9	2.9	5.7
v34	4.0	5.8	**3.5**	8.6
v35	**1.2**	**1.2**	**1.2**	5.7
v36	**1.2**	**1.2**	1.7	2.9
v37	2.9	**1.7**	2.3	8.6
v39	4.6	**3.5**	4.0	6.9
v41	**8.7**	12.7	9.2	**8.6**

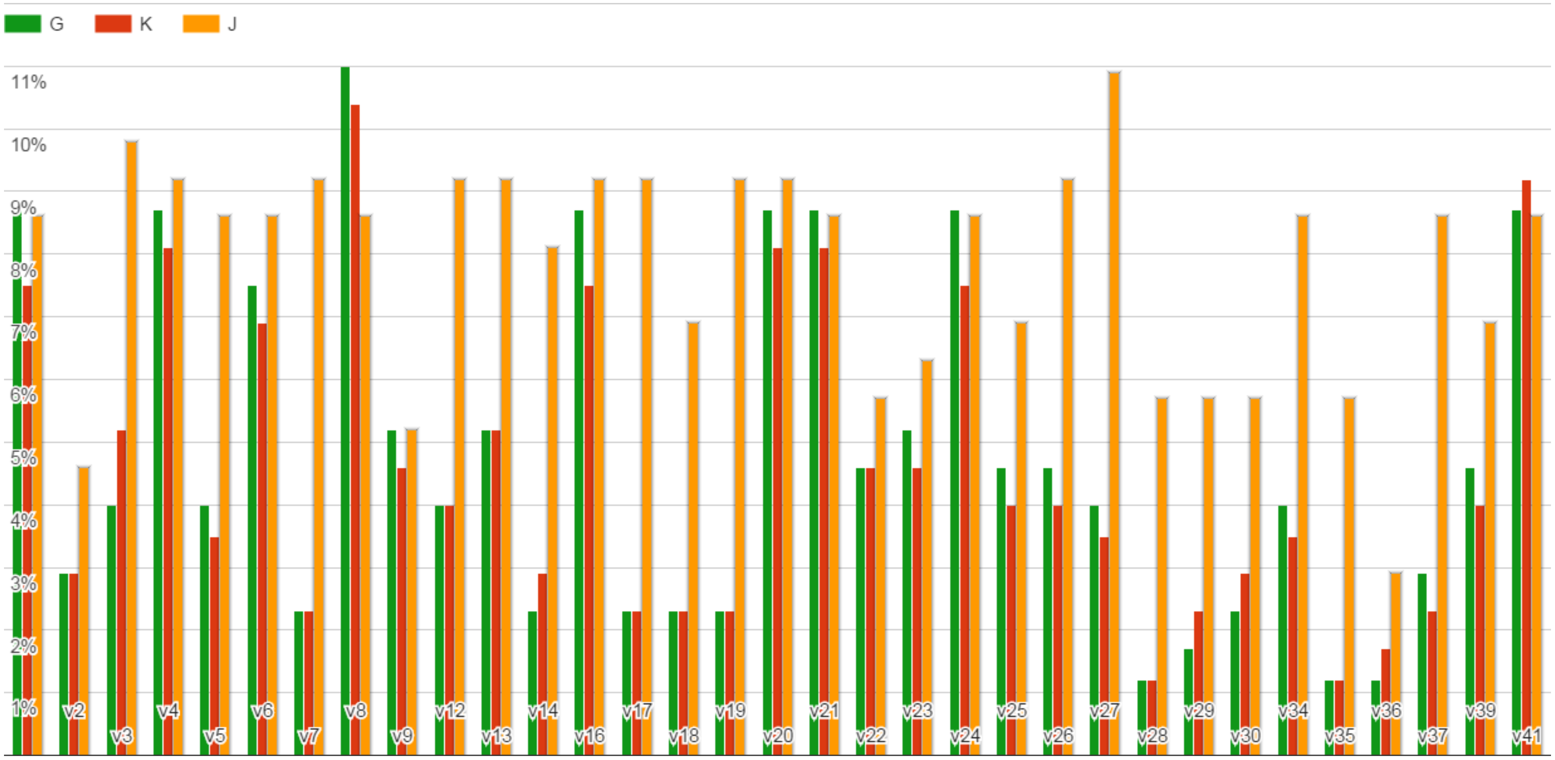
Table 2. Percentage of Lines of Code Returned by Localisation; see Table 1 for legend.

Tarantula: 19th returned location (**10.8%**)

"Optimal" Spectrum-Based: 17th ranked location (**9.9%**)

KLEE back-end: 4th ranked location (**2.3%**)

Localisation Performance



Localisation Performance

	G	N	K	J
v1	8.7	10.4	**7.5**	8.6
v2	**2.9**	**2.9**	**2.9**	4.6
v3	**4.0**	8.7	5.2	9.8
v4	8.7	11.0	**8.1**	9.2
v5	4.0	8.1	**3.5**	8.6
v6	7.5	11.0	**6.9**	8.6
v7	2.3	**1.7**	2.3	9.2
v8	11.0	16.8	10.4	**8.6**
v9	5.2	**4.6**	**4.6**	5.2
v12	**4.0**	4.6	**4.0**	9.2
v13	**5.2**	9.2	**5.2**	9.2
v14	**2.3**	2.9	2.9	8.1
v16	8.7	10.4	**7.5**	9.2
v17	2.3	**1.7**	2.3	9.2
v18	**2.3**	**2.3**	**2.3**	6.9
v19	2.3	**1.7**	2.3	9.2
v20	8.7	12.1	**8.1**	9.2
v21	8.7	12.7	**8.1**	8.6
v22	4.6	**4.0**	4.6	5.7
v23	5.2	**4.6**	**4.6**	6.3
v24	8.7	11.0	**7.5**	8.6
v25	4.6	**3.5**	4.0	6.9
v26	4.6	6.9	**4.0**	9.2
v27	4.0	8.1	**3.5**	10.9
v28	**1.2**	**1.2**	**1.2**	5.7
v29	**1.7**	2.3	2.3	5.7
v30	**2.3**	2.9	2.9	5.7
v34	4.0	5.8	**3.5**	8.6
v35	**1.2**	**1.2**	**1.2**	5.7
v36	**1.2**	**1.2**	1.7	2.9
v37	2.9	**1.7**	2.3	8.6
v39	4.6	**3.5**	4.0	6.9
v41	**8.7**	12.7	9.2	**8.6**

Table 2. Percentage of Lines of Code Returned by Localisation; see Table 1 for legend.

Tarantula: 19th ranked location (**10.8%**)

“Optimal” Spectrum-Based: 17th ranked location (**9.9%**)

KLEE back-end: 4th ranked location (**2.3%**)

Conclusion

- An **improved search** through the test suite.
- Generates **genuine lists of repair locations** that could be expressed as a look-up table **at any assignment**.
- **Time performance** in line with other model-based fault localisation techniques.
- **Outperforms** technique's originally published implementation and naive reimplementation by **more than two orders of magnitude**.
- **More consistent** than **localisation performance** of other techniques and without compromising the narrowing extent; avoids false negatives of the competition.

Future Work

- Can work on **lack of oracle** (known correct output for test cases), even mixed test suites with both types.
- Can **go beyond assignments** as localisation.
- Can do **more than single-fault assumption**.
- All based on limits of the **symbolic analyser** that underlies it so we **piggyback on their progress**.
- Currently working on **larger examples**, including C code from students.

References

Griesmayer, A., Staber, S., Bloem, R.: **Automated Fault Localization for C Programs**. Electronic Notes in Theoretical Computer Science, pp. 95–111 (2007)

Jose, M., Majumdar, R.: **Cause Clue Clauses: Error Localization Using Maximum Satisfiability**. SIGPLAN Not. 46(6), pp. 437–446 (2011)

Naish, L., Lee, H.J., Ramamohanarao, K.: **A Model for Spectra-based Software Diagnosis**. ACM Trans. Softw. Eng. Methodol. 20(3), 11A (2011)