Correctness Attraction: A Study of Stability of Software Behavior Under Runtime Perturbation

Benjamin Danglot, Philippe Preux, Benoit Baudry, **Martin Monperrus**

Inria, Univ. Lille & KTH Royal Institute of Technology

ICSE, Gothenburg, May 31 2018

Journal First, Empirical Software Engineering, 2017

Introduction

Dijkstra: In a program, *"the smallest possible perturbations – i.e. changes of a single bit – can have the most drastic consequences."*

On the Cruelty of Really Teaching Computer Science, Austin, 2 December 1988

To what extent is that correct?

Example: Normal Execution

```
public int function(int bound) {
    int acc = 0;
    int mask = 0x02;
    for (int i = bound ; i > 0 ; i--) {
        acc |= i >> mask;
    }
    return acc;
}
```

input: bound = 8

Iteration	acc	i
1	2	8
2	3	7
3	3	6
4	3	5
5	3	4
6	3	3
7	3	2
8	3	1

output: acc = 3

Example: Perturbed Execution

```
public int function(int bound) {
    int acc = 0;
    int mask = 0x02;
    for (int i = bound ; i > 0 ; i--) {
        acc |= p(i, 1) >> mask;
    }
    return acc;
}
```

input: $bound = 8$

Iteration	acc	i	acc	i	
1	2	8	2	8	
2	3	7	2	7	+1
3	3	6	3	6	
4	3	5	3	5	
7	3	2	3	2	
8	3	1	3	1	

output: $acc = 3$

Experimental Methodology

1. *Perturbation model:* a change on the program state.
   PONE: integer value is incremented.

Experimental Methodology

1. *Perturbation model:* a change on the program state.
   PONE: integer value is incremented.
2. List all x perturbation points (eg `return x`);

Experimental Methodology

1. *Perturbation model:* a change on the program state.
   PONE: integer value is incremented.
2. List all x perturbation points (eg `return x`);
3. Generate y inputs

Experimental Methodology

1. *Perturbation model:* a change on the program state.
   PONE: integer value is incremented.
2. List all x perturbation points (eg `return x`);
3. Generate y inputs
4. Explore exhaustively all possible perturbations
   $(\sum_x \sum_y \sum_{iterations})$

Experimental Methodology

1. *Perturbation model:* a change on the program state.
   PONE: integer value is incremented.
2. List all x perturbation points (eg `return x`);
3. Generate y inputs
4. Explore exhaustively all possible perturbations
   $(\sum_x \sum_y \sum_{iterations})$
5. Use a perfect oracle to assess correctness
   (eg `x == decompress(compress(x))`)

Example: 4950 Perturbed Executions

```java
public int function(int bound) {
    int acc = 0;
    int mask = 0x02;
    for (int i = bound ; i > 0 ; i--) {
        acc |= p(i, 1) >> mask;
    }
    return acc;
}
```

input: $bound = 8$

Iteration	acc	i	acc	i	
1	2	8	2	8	
2	3	7	2	7	+1
3	3	6	3	6	
4	3	5	3	5	
7	3	2	3	2	
8	3	1	3	1	

output: $acc = 3$

Example: exhaustive exploration

- Inputs : $bound \in [0; 100]$
- 4950 perturbed executions
- 99.90% correctness ratio
- 5 failed executions (0.1%)

Experimental Results

Experiment: PONE, 20 inputs

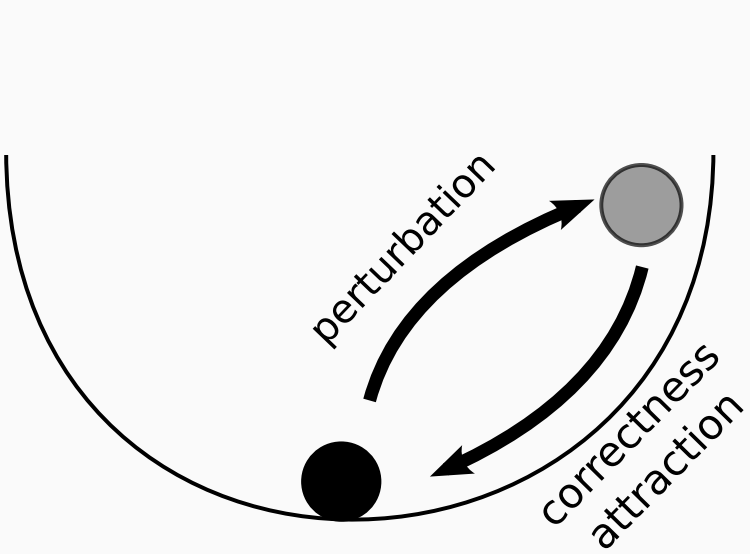
Subject	N_{pp}^{int}	—Search space—	correctness ratio
md5	164	237680	— 29.67 %
rc4	115	165140	—— 38.04 %
linreg	75	543720	——— 47.88 %
rsa	117	2576	———— 54.97 %
sudoku	89	98211	———— 68.8 %
zip	19	38840	————— 76.09 %
quicksort	41	151444	————— 77.6 %
lcs	79	231786	————— 89.93 %
laguerre	72	423454	————— 90.64 %
canny	450	616161	————— 94.55 %
total	1221	2509012	———— 66.817 %

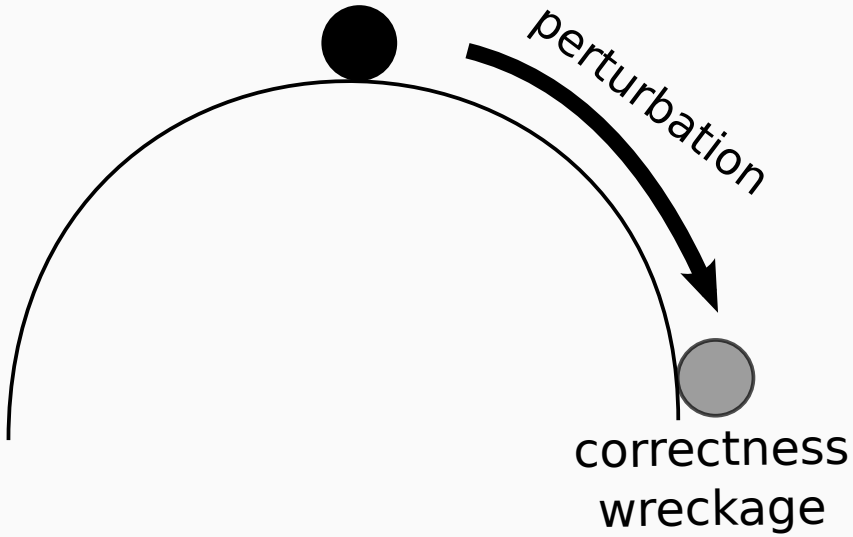
Answer to RQ

RQ: How does software behave under perturbation?

- In 1676446 / 2509012 (66%) perturbed executions, the final output is fully correct.

Concept: Correctness Attraction





(inspired from “attraction basin” in physics)

Novelty: methodology

- Not a bit-flip perturbation model ($\neq$ fault injection)
- Not a permanent code modification ($\neq$ mutation)
- Exhaustive exploration of the perturbation space ($\neq$ random search)
- Use of perfect oracle (not only the absence of crashes, $\neq$ approximate computing)

Novelty: results

Confirms anecdotally reported results on:

- Silent / benign faults;
- Unkillable mutants;
- The stability of software systems in production.

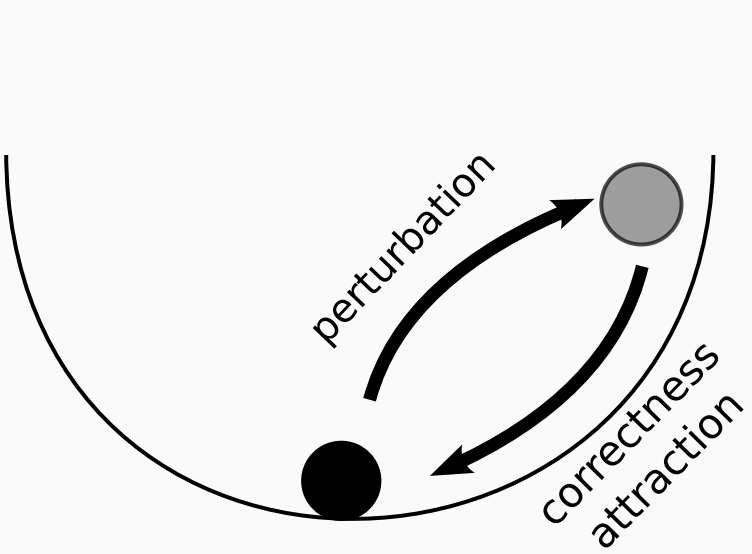
But 1) at a larger scale, 2) with a perfect oracle
3) with a name: “correctness attraction”

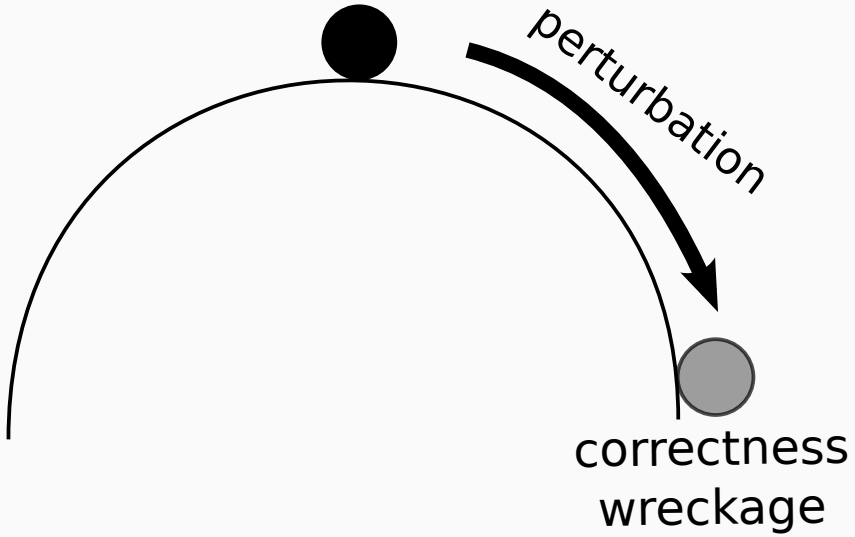
Taxonomy

Manual analysis of cases to create a taxonomy of 7 reasons of correctness attraction:

- Potential alternative executions (inherent redundancy)
- Fixed point effect
- Extra resources & slack
- + 4 others, see paper.

Conclusion: Correctness Attraction





In 1676446 / 2509012 (66%) perturbed executions, the final output is fully correct.

Major applications in software security.