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

Benjamin DANGLOT

December 6^{th}, 2017

benjamin.danglot@inria.fr

Kungliga Tekniska Högskolan (KTH) - European Chaos Engineering Day

Introduction

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

Goal: Explore The Perturbability Envelop of Software

RQ: How does Software behave under perturbation?

Goal: Explore The Perturbability Envelop of Software

RQ: How does Software behave under perturbation?

Perturbation is a change that occurs runtime ($\neq$ mutant).

Goal: Explore The Perturbability Envelop of Software

RQ: How does Software behave under perturbation?

Perturbation is a change that occurs runtime ($\neq$ mutant).

Attract Protocol: Explore the perturbability of Software:

Goal: Explore The Perturbability Envelop of Software

RQ: How does Software behave under perturbation?

Perturbation is a change that occurs runtime ($\neq$ mutant).

Attract Protocol: Explore the perturbability of Software:

- Exploring exhaustively

Goal: Explore The Perturbability Envelop of Software

RQ: How does Software behave under perturbation?

Perturbation is a change that occurs runtime ($\neq$ mutant).

Attract Protocol: Explore the perturbability of Software:

- Exploring exhaustively
- Using perfect oracle

The Attract Protocol

The Attract Protocol: Example

```
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

The Attract Protocol: Example

```
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

The Attract Protocol: Perturbation Points

```
acc |= i >> mask;
```

The Attract Protocol: Perturbation Points

```
acc |= i >> mask;
```

```
acc |= i >> mask;
```

The Attract Protocol: Perturbation Points

```
acc |= i >> mask;
```

```
acc |= i >> mask;
```

```
acc |= p(i, 1) >> mask;
```

The Attract Protocol: Perturbation Points

```
acc |= i >> mask;
```

```
acc |= i >> mask;
```

```
acc |= p(i, 1) >> mask;
```

```
public int p(int integer, int id) {
    if (mustBePerturbed(id)) {
        return integer + 1;
    } else {
        return integer;
    }
}
```

The Attract Protocol: 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	
5	3	4	3	4	
6	3	3	3	3	
7	3	2	3	2	
8	3	1	3	1	

output: $acc = 3$

The Attract Protocol: 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	
5	3	4	3	4	
6	3	3	3	3	
7	3	2	3	2	
8	3	1	3	1	

output: $acc = 3$

The Attract Protocol: Applied On The Example

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

The Attract Protocol: Exhaustive Exploration

Reference Execution

It.	*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: $acc = 3$

Perturbed Execution 1

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

output: $acc = 3$

Perturbed Execution 2

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

output: $acc = 3$

Perturbed Execution 5

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

output: $acc = 1$

The Attract Protocol: Perfect Oracle

Reference Execution

It.	*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: $acc = 3$

Perturbed Execution 1

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

output: $acc = 3$

Perturbed Execution 2

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

output: $acc = 3$

Perturbed Execution 5

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

output: $acc = 1$

The Attract Protocol: Core Algorithm

```
1  instrument(prog);
2  for each input i in I do
3      n[pp, i] ← runWithoutPerturbation(prog, i) ∀pp ∈ prog;
4      for each perturbation point pp in prog do
5          for j = 0, to n[pp, i] do
6              o ← runWithPerturbationAt(prog, i, pp, j);
7              if oracle.assert(i, o) then
8                  success ← success + 1;
9              else
10                 failure ← failure + 1;
11             end
12         end
13     end
14 end
```

Experiment

Experiment: PONE

$+1$ on every integer expression for each call of each perturbation point

Experiment: PONE

+1 on every integer expression for each call of each perturbation point

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

Answer

RQ: How does Software behave under perturbation?

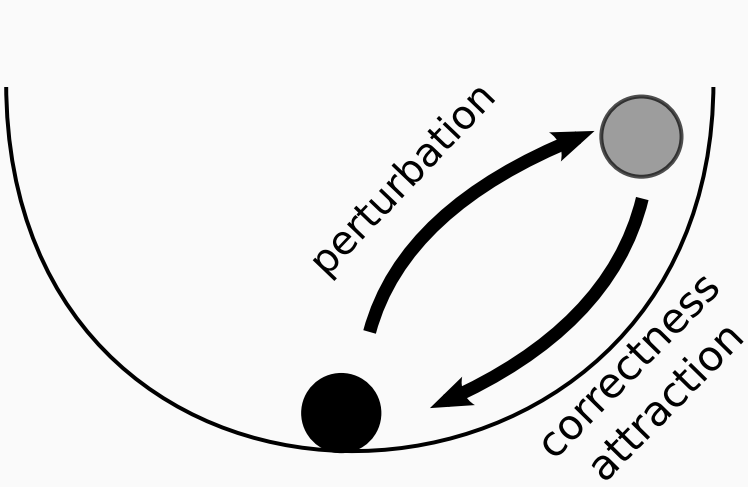
Answer

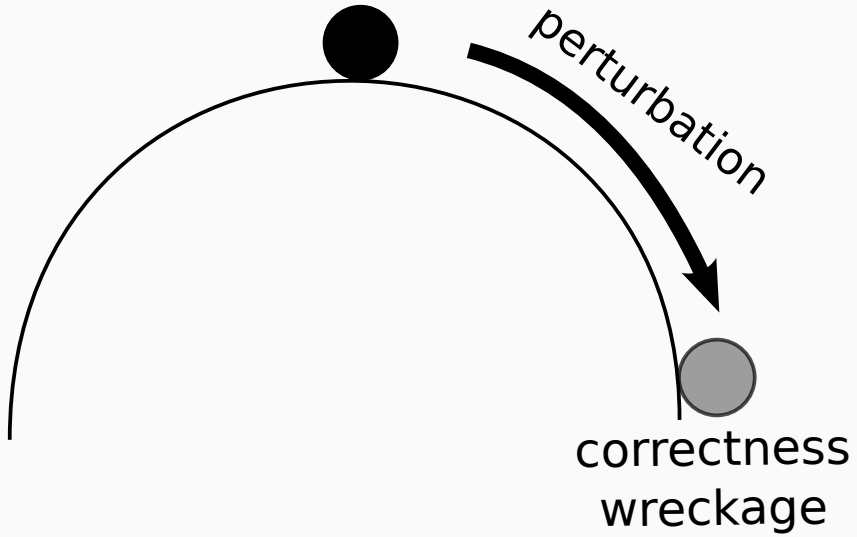
RQ: How does Software behave under perturbation?

Answer:

- In 1676446(66.817%) of 2509012 perturbed executions, the final output is perfectly correct.
- Software are able to recover from perturbation, and produce the perfect output.

Answer: Correctness Attraction





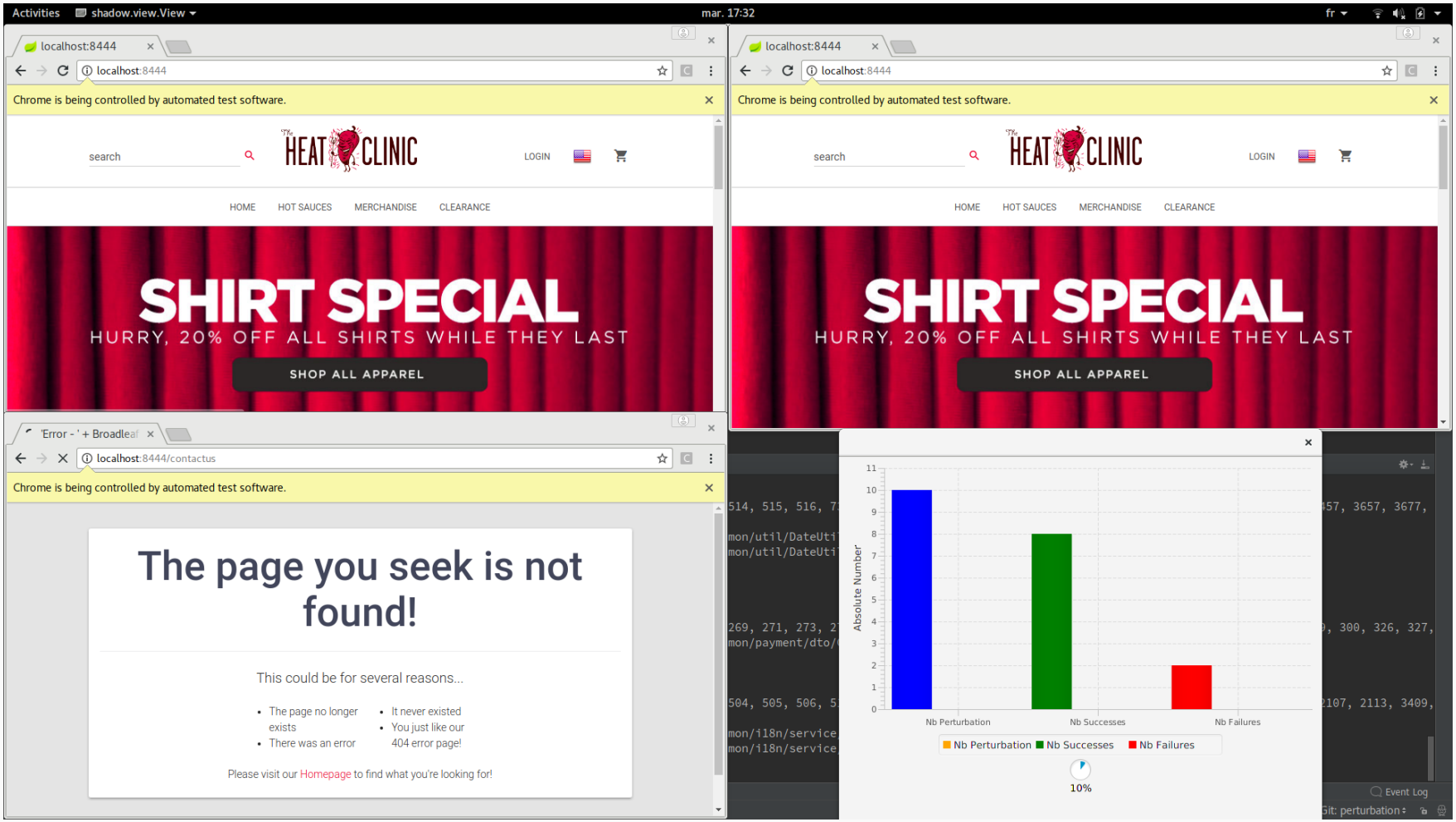
Demo

Demo: What it does?

First fine-grained chaos inside a “production” application.

- Explore the perturbability of a web application
- Running an instrumented version of a e-commerce app
- At each request, one perturbation point is enabled
- The perturbation is $+1$ on integer expressions

Demo: screen


Activities
shadow.view.View
mar. 17:32
fr
localhost:8444
Chrome is being controlled by automated test software.
search
THE HEAT CLINIC
LOGIN
HOME
HOT SAUCES
MERCHANDISE
CLEARANCE
SHIRT SPECIAL
HURRY, 20% OFF ALL SHIRTS WHILE THEY LAST
SHOP ALL APPAREL
'Error - ' + Broadleaf
localhost:8444/contactus
The page you seek is not found!
This could be for several reasons...
The page no longer exists
There was an error
It never existed
You just like our 404 error page!
Please visit our Homepage to find what you're looking for!
Absolute Number
Nb Perturbation
Nb Successes
Nb Failures
10%
Event Log

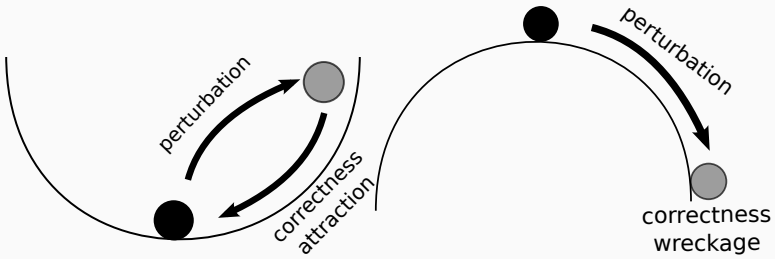
Future Works

Future Works

- **Future RQ1:** How does the "strength" of the perturbation impact the correctness ratio: *i.e.* $+1000$ instead of $+1$, or 50% of probability to enable each perturbation point? $\Rightarrow$ increase the Chaos.
- **Future RQ2:** How does "production" Software behave under the Attract protocol?
- **Future RQ3:** Does the correctness ratio can be used as proxy to measure the "Antifragility" / "Resiliency" of Software?
- **Future RQ4:** Could we engineer our Software to increase the Correctness and so increase the "Resiliency"?

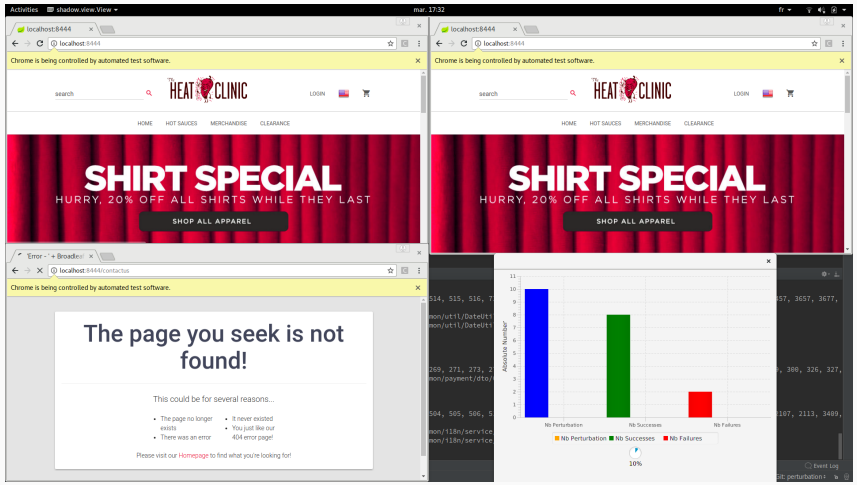
Conclusion

Conclusion



- Exhaustive Exploration
- Perfect Oracle

Subject correctness ratio	
quicksort	——— 77.6 %
zip	——— 76.09 %
sudoku	——— 68.8 %
md5	— 29.67 %
rsa	——— 54.97 %
rc4	—— 38.04 %
canny	——— 94.55 %
lcs	——— 89.93 %
laguerre	——— 90.64 %
linreg	—— 47.88 %
total	——— 66.817 %



Correctness VS Approximate Computing

Approximate Computing accepts light degradation of the output

VS

Correctness expects a **PERFECT** output

Taxonomy

Taxonomy of 7 reasons of Correctness Attraction:

- Natural randomness
- Relaxed problem
- Nullified perturbation
- Overfit to input data
- Potential alternative executions
- Fixed point effect
- Extra resources

Threats To Validity

Internal threats

- Bugs in the implementation

External threats

- Limited dataset: 10 projects, only Java
- Overfitting on input values