Phuzzer: Random(-ish) Testing for Pharo

ESUG - 2023 - Lyon

guillermo.polito@inria.fr
@guillep

* Supported by AlaMVic Action Exploratoire INRIA
First: About Me

• **Keywords:** compilers, testing, test generation

• **Ph.D.:** Reflection, debloating, dynamic updates

• **Interests:** tooling, benchmarking, 日本語, board games, concurrency

Talk to me!

Or: guillermo.polito@inria.fr
The Essence of Testing

“Program testing can be used to show the presence of bugs, but never to show their absence”

- E. Dijkstra
What is a Good Test?

“A good test is a test that catches bugs”

- me
1988, Barton Miller observed random crashes in UNIX utilities. 

Shouldn’t it be more robust?
An Empirical Study on the Reliability of UNIX Utilities

- Call utilities with *random* inputs
- ~90 tested utilities
- 25-33% crashed

```
fuzz 100000 -o outfile | deqn
```
Random Fuzzer

(1 to: 10) collect: [ :e | PzRandomFuzzer new fuzz ]
Let's Test some Parser

f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.

- Pharo 11
- String>>asDate

PASS "7 February 2039"
FAIL "DateError: day is after month ends"
PASS "1 June 2002"
PASS "5 August 13836"
FAIL "DateError: day may not be zero or negative"
PASS "1 January 2004"
FAIL "#isAlphaNumeric was sent to nil"
FAIL "SubscriptOutOfBounds: 0"
PASS "7 June 2009"
PASS "3 April 2001"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 72"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day is after month ends"
PASS "3 April 1991"
PASS "3 October 2001"
FAIL "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 0"
Let’s Test some Parser

f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.

• Pharo 11
• String>>asDate

• 12/20 = 60% of failures?
Refining the Results

- Pharo 11
- String>>asDate
- DateError is an expected error!

```small
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
r expectedException: DateError.
f run: r times: 20.
```
Refining the Results

- **Pharo 11**
- **String>>asDate**
- **DateError is an expected error!**
- **4/20 = 5% of errors**

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [:e | e asDate].
r expectedException: DateError.
f run: r times: 20.
```
Changing the Input

“Large charset”
\f := PzRandomFuzzer new.
\f charStart: Character tab.
\f charRange: 500.
(1 to: 10) collect: [ :e | \f fuzz ]

“Alphanumeric”
\f := PzRandomFuzzer new.
\f charStart: $A.
\f charRange: 50.
(1 to: 10) collect: [ :e | \f fuzz ]
Fuzzer ratio over 100 runs

\[ r := \text{PzBlockRunner} \text{ on: } [ :e \mid e \text{ asDate} ]. \]
\[ r \text{ expectedException: DateError.} \]
\[ f \text{ run: } r \text{ times: 100.} \]

<table>
<thead>
<tr>
<th>Fuzzer</th>
<th>Pass</th>
<th>Expected Fail</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weird Chars</td>
<td>49 %</td>
<td>29 %</td>
<td>22 %</td>
</tr>
<tr>
<td>Large Char set</td>
<td>0 %</td>
<td>0 %</td>
<td>100 %</td>
</tr>
<tr>
<td>Alphanumeric</td>
<td>0 %</td>
<td>0 %</td>
<td>100 %</td>
</tr>
</tbody>
</table>
Random Inputs Fail Easily

• We could expect to break something with fully random inputs
• This could be solved with input sanitizing

• What if we have almost correct inputs?
• Looks like a date, quacks like a date, parses as a date?
We need to generate syntactically and semantically valid inputs
We need to generate **syntactically** and **semantically** valid inputs
Date Fuzzer

(1 to: 10) collect: [ :e | PzDateFuzzer new fuzz ]

23 5
7/February-6
7,February0
0/february/7
9 february 0
7 February-9
February 0,1
4/February,4
february/0 7
1January,8
Grammars as Input Descriptions

• Grammars describe languages

• Usually used for parsing purposes, but…

• Key idea (from the 60s) => structured fuzzing with grammars
Date Grammar

ntNumber --> ntDigit, ntNumber | ntDigit.
ntDigit --> ($0 - $9).

ntDate
  --> ntDay, ntSeparator, ntMonth, ntSeparator, ntYear
  | ntMonth, ntSeparator, ntDay, ntSeparator, ntYear
  | ntYear, ntSeparator, ntMonth, ntSeparator, ntDay.
ntSeparator --> '' | ' ' | '-' | ',' | '/'.
ntDay --> ntNumber.
ntMonth
  --> ntNumber
  | 'january' | 'January'
  | 'february' | 'February'.
ntYear --> ntNumber.
Grammar Fuzzer

(1 to: 10) collect: [ :e | (PzGrammarFuzzer on: PzDateGrammar new) fuzz ]

23 5
7/February-6
7,February0
0/february/7
9 february 0
7 February-9
February 0,1
4/February,4
february/0 7
1January,8
Let’s test some parser (bis)

\[ f := \text{PzGrammarFuzzer} \text{ on: PzDateGrammar new.} \]
\[ r := \text{PzBlockRunner} \text{ on: [ :e | e asDate ].} \]
\[ r \text{ expectedException: DateError.} \]
\[ f \text{ run: r times: 100.} \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>81 %</td>
</tr>
<tr>
<td>Expected-Fail</td>
<td>10 %</td>
</tr>
<tr>
<td>Fail</td>
<td>9 %</td>
</tr>
</tbody>
</table>

- Simple Date grammar fuzzing has a high success ratio
Looking at the bugs

- Out of 135 bugs fuzzing 1000 cases

<table>
<thead>
<tr>
<th>Issue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>method not understood during parsing</td>
<td>83 %</td>
</tr>
<tr>
<td>Out of bounds during parsing</td>
<td>13 %</td>
</tr>
<tr>
<td>Validation with generic error during parsing</td>
<td>4 %</td>
</tr>
</tbody>
</table>
Back-tracking a Bit

f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.

• Pharo 11
• String>>asDate

PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "DateError: day may not be zero or negative"
PASS "5 January 0228"
PASS "DateError: day may not be zero or negative"
PASS "7 September 1996"
PASS "2 January 2008"
Back-tracking a Bit

f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.

• Pharo 11
• String>>asDate

PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "5 3"
PASS "DateError: day is after month ends"
PASS "7 9"
PASS "2 3"

How do we decide: what is a PASS, what is a FAIL?
When Dates Should Parse

• DateError is an expected error
• Malformed inputs should fail!
  • .+!;/.*52"%?3720("/")!*43,"4@")>'(,"0(+7?
  • ;% *:(41)215>/1890}@ 3"@3.35+6

f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.

PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
PASS "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASSE "Date"
PASS "5 3"
PASS "Date"
PASSE "7 9"
PASSE "2 3"

How do we decide: what is a PASS, what is a FAIL?
The Oracle Problem

Given a program and an input,

How can we distinguish correct from incorrect behavior?
The Oracle Problem

Given a program and an input,

How can we **automatically** distinguish correct from incorrect behavior?
Remember Assertions

SetTest >> testAdd

| aSet |
"Context"
aSet := Set new.

"Stimuli"
aSet add: 5.
aSet add: 5.

"Check"
self assert: aSet size equals: 1.

in this context
when this happens
then this should happen
Comparisons against known values

SetTest >> testAdd

| aSet | "Context"
|------|---------------
| "Stimuli"
| aSet add: 5.
| aSet add: 5.

"Check"
| self assert: aSet size equals 1.

**in this context**
**when this happens**
**then this should happen**
Assertions against similar values?

SetTest >> testAdd

| aSet | "Context"
aSet := Set new.

"Stimuli"
| aSet add: 5.
aSet add: 5.

"Check"
| self assert: aSet size equals ????.

in this context
when this happens
then this should happen
Can we use another parser?

SetTest >> testAdd

| aSet | "Context"
|------|-----------------
| aSet := Set new.

"Stimuli"
| aSet add: 5.
| aSet add: 5.

"Check"
| self assert: aSet size equals ?????.

• (dis)agreement is evidence!
• Agreement: parsers have same behavior
• Disagreement: is there a bug?
Date>>fromString vs DateParser

• DateParser is a structured parser based on a specified format

  DateParser readFrom: string readStream pattern: ‘m-d-yyyy’

• Should be more strict than asDate

• What kind of comparisons are fair/safe/legal?
A m-d-yyy Grammar

ntNumber  -->  ntDigit, ntNumber | ntDigit.
ntDigit  -->  ($0 - $9).

ntDate
  -->  ntMonth, ntSeparator, ntDay, ntSeparator, ntYear.
ntSeparator  -->  '-'.
ntDay  -->  ntDigit | '1', ntDigit | '2', ntDigit | '30' | '31'.
ntMonth  -->  ntDigit | '11' | '12'.
ntYear  -->  ntDigit, ntDigit, ntDigit, ntDigit, ntDigit.
Differential Parser Testing

runnerA := PzBlockRunner on: [ :e | e asDate ]. runnerA expectedException: DateError.
runnerB := PzBlockRunner on: [ :e | (Date readFrom: e readStream pattern: 'm-d-yyyy') ]. runnerB expectedException: DateError.
diffRunner := PzDifferentialRunner new
  runnerA: runnerA;
  runnerB: runnerB;
  yourself.

f := PzGrammarFuzzer on: PzDateMDYYYYGrammar new. f run: diffRunner times: 100.
Results

• fuzz 100 times
• 3/100 errors!
## Results

- fuzz 100 times
- 3/100 errors!
- Dates are mostly correct!
We need to generate **syntactically** and semantically valid inputs
We need to generate **syntactically** and semantically valid inputs

**But Slightly Wrong**
Mutations as Fuzzers

E.g., change one + by a -
Random String Mutator

```ruby
f := PzMutationFuzzer new
seed: { ‘abcd’ };
yourself.

(1 to: 10) collect: [ :e | f fuzz ]
```

3ou
AbC|dM
aEbcN`
bca`c$#
bcc
abc$
aabcd
!cbb~d
;
String Mutations

- **Insert** a *random* character in a *random position* of the string
- **Delete** a character in a *random position* of the string
- **Replace** a character by a *random* character in a *random position* of the string
Chaining Fuzzers

• Mutating a correct value
  • pre-existent or grammar-fuzzed
  • produces *probably* correct values
  • and *probably incorrect* too
Results by crash location
(signaler context method + pc)

• 68% disagreements - 6826 out of 10k fuzzings
• new errors!

<table>
<thead>
<tr>
<th>Error</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream still with data</td>
<td>2847</td>
<td>41,71 %</td>
</tr>
<tr>
<td>Input doesn’t match pattern</td>
<td>1534</td>
<td>22,47 %</td>
</tr>
<tr>
<td>Wrong year digits</td>
<td>1379</td>
<td>20,20 %</td>
</tr>
<tr>
<td>Wrong day/month</td>
<td>654</td>
<td>9,58   %</td>
</tr>
<tr>
<td>No error!</td>
<td>50</td>
<td>0,73   %</td>
</tr>
<tr>
<td>Day after month end</td>
<td>9</td>
<td>0,13   %</td>
</tr>
<tr>
<td>Day zero or negative</td>
<td>5</td>
<td>0,07   %</td>
</tr>
</tbody>
</table>
Results by crash location
(signaler context method + pc)

- 68% disagreements - 6826 out of 10k fuzzings
- new errors!

<table>
<thead>
<tr>
<th>Error</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream still with data</td>
<td>2847</td>
<td>41,71 %</td>
</tr>
<tr>
<td>Input doesn’t match pattern</td>
<td>1534</td>
<td>22,47 %</td>
</tr>
<tr>
<td>Wrong year digits</td>
<td>1379</td>
<td>20,20 %</td>
</tr>
<tr>
<td>Wrong day/month</td>
<td>854</td>
<td>9,58 %</td>
</tr>
<tr>
<td>No error!</td>
<td>50</td>
<td>0,73 %</td>
</tr>
<tr>
<td>Day after month end</td>
<td>5</td>
<td>0,13 %</td>
</tr>
<tr>
<td>Day zero or negative</td>
<td>5</td>
<td>0,07 %</td>
</tr>
</tbody>
</table>
When asDate was ok (50/10000)

'10-30-s9 6'

DateParser

asDate

Result NOK

Result OK
When asDate was ok (50/10000)

DateParser accepts (and ignores) non numeric characters in year
Takeaways

• Simple random inputs can unveil bugs
  • but, random inputs get random outputs!

• Adding some domain knowledge makes fuzzing more efficient
  • grammars, mutations, expected exceptions…

• Two programs following the same specification can test each other
  • Yet, maybe neither holds the ground truth

https://github.com/Alamvic/phuzzer
https://github.com/Alamvic/gnocco

* Supported by AlaMVic Action Exploratoire INRIA
Heap Fuzzing

- Allocations
  - where: what memory region?
  - kind: normal object, array?
  - size: how many slots?
  - root?
- Mutations: \texttt{obj1.a = obj2}
- Garbage Collection Events
Interpreter-Guided Compiler Testing


```plaintext
Interpreter >> bytecodePrimAdd
| rcvr arg result |
rcvr := self internalStackValue: 1.
arg := self internalStackValue: 0.
(objectMemory areIntegers: rcvr and: arg) ifTrue: [
    result := (objectMemory integerValueOf: rcvr) + (objectMemory integerValueOf: arg).
    "Check for overflow"
    (objectMemory isIntegerValueOf: result) ifTrue: [
        self internalPop: 2
        thenPush: (objectMemory integerObjectValue: result)
        ^ self fetchNextBytecode "success"
    ].
    "Slow path, message send"
    self normalSend

... # previous bytecode IR
  checkSmallInteger t0
  jumpzero notsmi
  checkSmallInteger t1
  jumpzero notsmi
  t2 := t0 + t1
  jumplfNotOverflow continue
  notsmi: #slow case first send
```

Interpreter-Guided Differential JIT Compiler Testing

Guillermo Polito
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRISTAL, F-59000 Lille, France
guillermo.polito@univ-lille.fr

Stéphane Ducasse
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRISTAL, France
stephane.ducasse@inria.fr

Pablo Teson
Pharo Consortium
Univ. Lille, Inria, CNRS, UMR 9189 CRISTAL, France
pteson@inria.fr

PLDI'22
Material

• The Fuzzing Book. Fuzzer Chapter. A. Zeller et al

• Fuzzing – Brute Force Vulnerability Discovery.

• Fuzzing for Software Security and Quality Assurance.
  Takanen et al. 2018

• An Empirical Study of the Reliability of UNIX Utilities
  Miller et al. Communications of the ACM’90

• Fuzz project assignment
Material

• The Fuzzing Book. Grammars Chapter. A. Zeller et al

• Gnocco
  https://github.com/Alamvic/gnocco/
Material

Material

- The Fuzzing Book. Mutation Chapter. A. Zeller et al
  https://www.fuzzingbook.org/html/MutationFuzzer.html

- Binary Fuzzing Strategies in AFL — Blog
Building a Random Fuzzer

- Choose a random size
- Choose random chars in a range
- Build up a string
- + sensible default values

```
PzRandomFuzzer >> fuzz | stringLength |
  stringLength := random
  nextIntegerBetween: minLength
  and: maxLength + 1.
```

```
^ String streamContents: [ : str |
  stringLength timesRepeat: [ 
    str nextPut: (random
      nextIntegerBetween: charStart asciiValue
      and: charStart asciiValue + charRange)
      asCharacter ] ]
```
Analysis II

- Some inputs PASS but do not respect the contract

"Answer an instance of created from a string with format mm.dd.yyyy or mm-dd-yyyy or mm/dd/yyyy"

'(?(2/-@=@:4?/(3$3(8"&,-2/&6&' asDate.

>> 4 February 2003

- Parser is too permissive
- Our runner is too permissive too => we should detect this as an error!
Building a Runner

PzRunner>>value: input

| result |
[ result := self basicRunOn: input ]
on: Error
do: [ :err |
  (expectedException notNil
   and: [ expectedException handles: err ])
   ifTrue: [ ^ self expectedFailureWith: { input . err freeze} ].
^ self failureWith: { input . err freeze} ].
^ self successWith: { input . result}
Building a Grammar Fuzzer

- Example, a number grammar

\[
\text{ntNumber} \rightarrow \text{ntDigit}, \text{ntNumber} \mid \text{ntDigit}.
\]
\[
\text{ntDigit} \rightarrow (\text{\$0 - \$9}).
\]
Desugarising into simple rules

- Example, a number grammar

\[
\begin{align*}
\text{ntNumber} & \rightarrow \text{ntDigit},\ \text{ntNumber} \mid \text{ntDigit}. \\
\text{ntDigit} & \rightarrow (\$0 - \$9).
\end{align*}
\]

\[
\begin{align*}
\text{ntNumber} & \rightarrow \text{ntDigit},\ \text{ntNumber} \\
\text{ntNumber} & \rightarrow \text{ntDigit}. \\
\text{ntDigit} & \rightarrow 0. \\
\text{ntDigit} & \rightarrow 1. \\
\ldots \\
\text{ntDigit} & \rightarrow 8. \\
\text{ntDigit} & \rightarrow 9.
\end{align*}
\]
Modelling as a Composite Pattern

- Example, a number grammar

\[
\begin{align*}
\text{ntNumber} &\rightarrow \text{ntDigit}, \text{ntNumber} \\
\text{ntNumber} &\rightarrow \text{ntDigit}. \\
\text{ntDigit} &\rightarrow 0. \\
\text{ntDigit} &\rightarrow 1. \\
\vdots \\
\text{ntDigit} &\rightarrow 8. \\
\text{ntDigit} &\rightarrow 9.
\end{align*}
\]
Instantiating the Model

• Example, a number grammar

ntNumber --> ntDigit, ntNumber
ntNumber --> ntDigit.
ntDigit --> 0.
ntDigit --> 1.
...
ntDigit --> 8.
ntDigit --> 9.
Building a Differential Runner

PzDifferentialRunner>>value: input

| resultA resultB |
resultA := self runnerA value: input.
resultB := self runnerB value: input.

resultA first = resultB first ifTrue: [
  ^ self successWith: { input . resultA . resultB } ].
^ self failureWith: { input . resultA . resultB}
Implementing a Mutation

```smalltalk
PzDeleteCharacterMutation>>mutate: aString

| index |
index := aString size atRandom.

^ (aString copyFrom: 1 to: index - 1),
  (aString copyFrom: index + 1 to: aString size)
```
Building a String Mutation Fuzzer

PzMutationFuzzer>>fuzz

| mutationCandidate trials |
mutationCandidate := seed at: (random nextInteger: seed size).
trials := random nextIntegerBetween: minMutations and: maxMutations.
trials timesRepeat: [
    mutationCandidate := self mutate: mutationCandidate ].
^ mutationCandidate

PzMutationFuzzer>>mutate: mutationCandidate
| mutationIndex mutation |
mutationIndex := random nextInteger: mutations size.
mutation := mutations at: mutationIndex.
^ mutation mutate: mutationCandidate
Domain-specific mutations

- E.g., swap day and month
  
  ```
  f := PzMutationFuzzer new
  seed: { '00-11-22' };
  mutations: { PzDayMonthSwapMutation new }
  yourself.
  ```

- E.g., change the schema of a URL (http by ftp)

- E.g., change the a smic operator by another (+ by -)
Remember Mutation Analysis

original program

Apply Mutation

mutant

Run Tests

mutated program

E.g., change one + by a -

Survived

Killed
Mutation Analysis vs Mutation Fuzzing

- **Mutation analysis** evaluates test suite *quality*
  - High Mutation Score => good tests
  - Surviving mutants => show missing tests, or are equivalent

- **Mutation fuzzing** creates small variants
  - There is no notion of score
  - Equivalent mutants could be valuable!