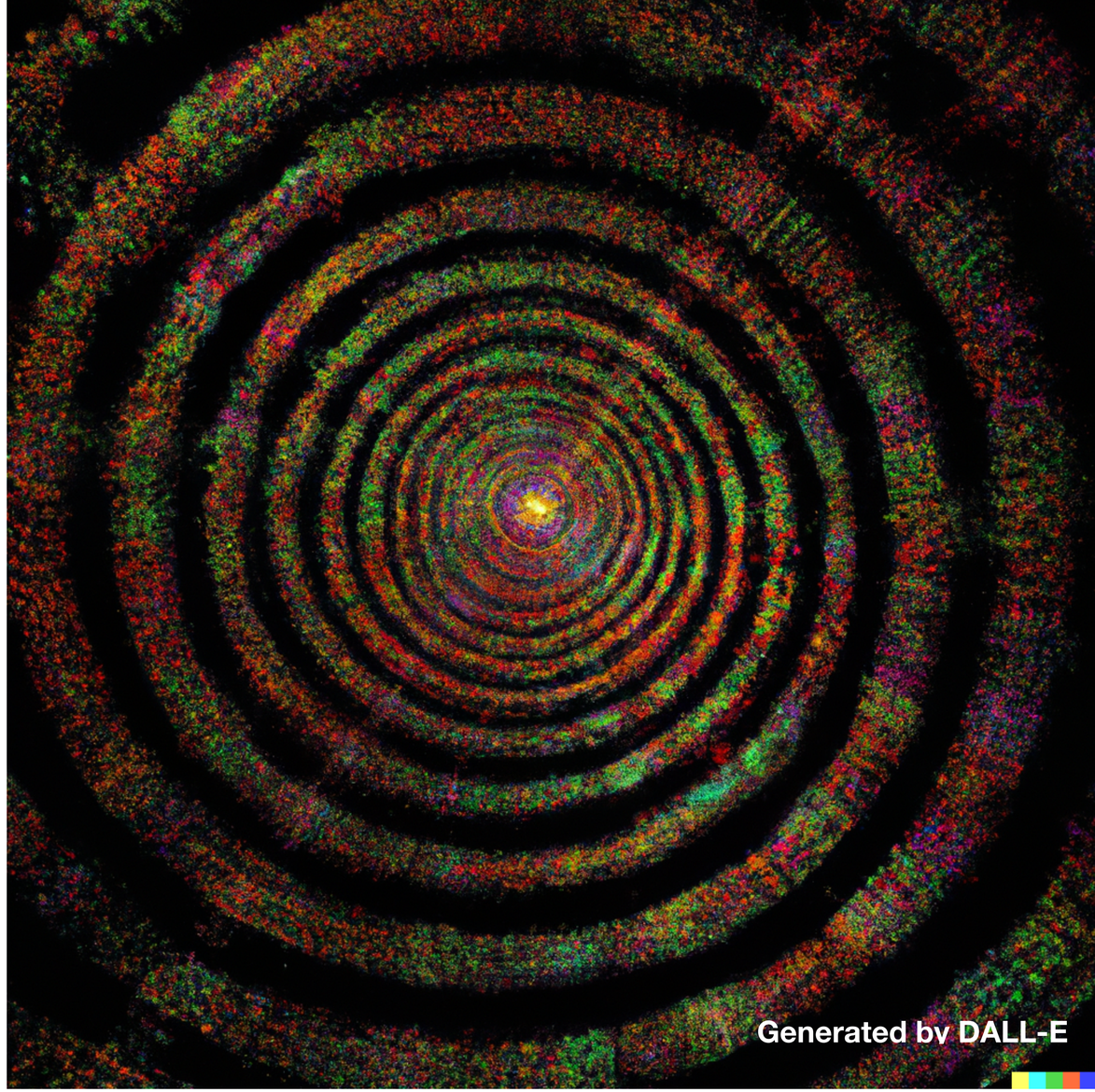
Phuzzer:
Random(-ish) Testing for Pharo

ESUG - 2023 - Lyon

guillermo.polito@inria.fr
@guillep



* Supported by AlaMVic Action Exploratoire INRIA

First: About Me

guillermo.polito@inria.fr
@guillep

- **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”

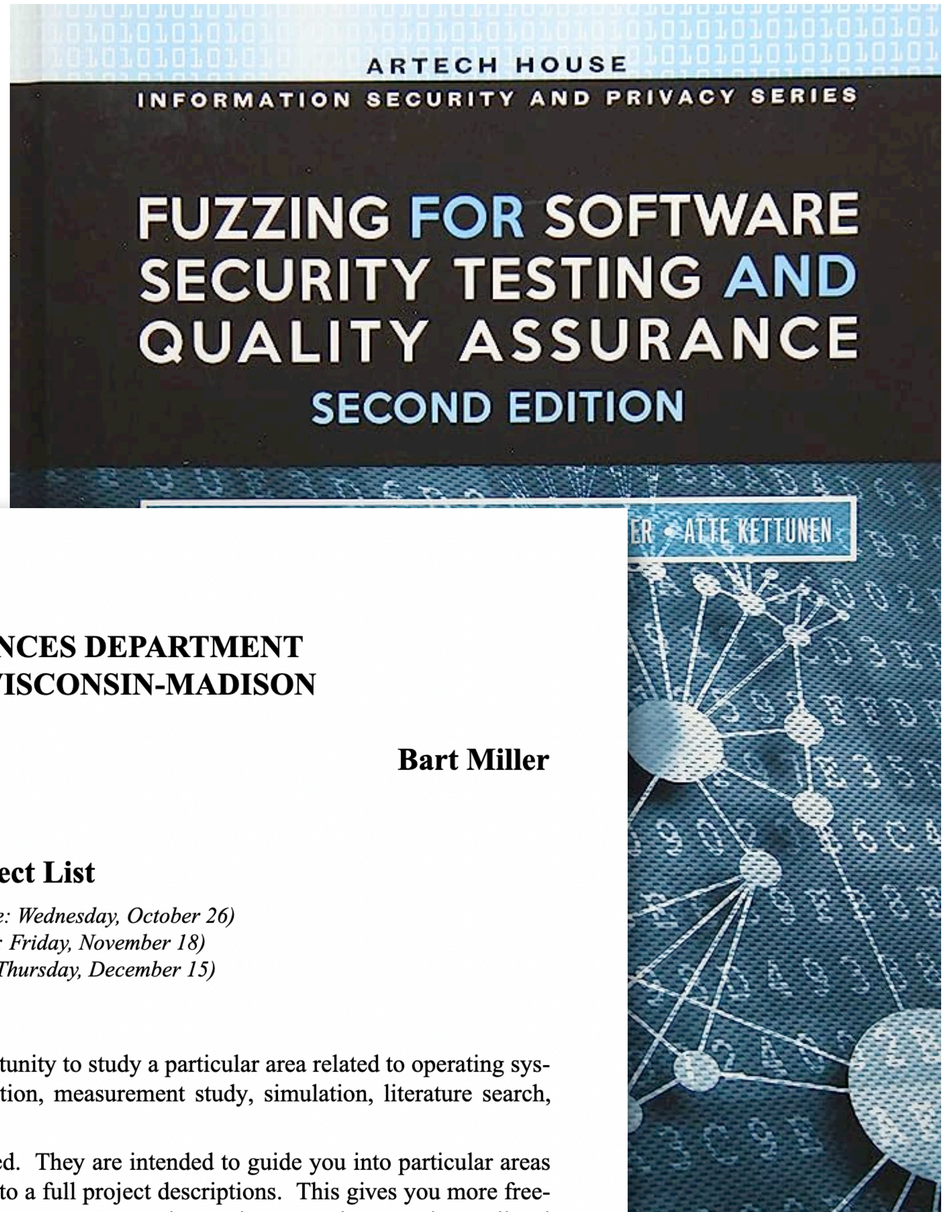
- me

The Fuzz Generator

1988, Barton Miller observed random

crashes in UNIX utilities.

Shouldn't it be more robust?



COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

CS 736
Fall 1988

Bart Miller

Project List

(Brief Description Due: Wednesday, October 26)
(Midway Interview: Friday, November 18)
(Final Report Due: Thursday, December 15)

General Comments

The projects are intended to give you an opportunity to study a particular area related to operating systems. Your project may require a test implementation, measurement study, simulation, literature search, paper design, or some combination of these.

The project suggestions below are briefly stated. They are intended to guide you into particular areas and you are expected to expand these suggestions into a full project descriptions. This gives you more free-

An Empirical Study on the Reliability of UNIX Utilities

- Call utilities with *random* inputs

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

- ~90 tested utilities
- **25-33% crashed**

Communications of the ACM'90

An Empirical Study of the Reliability

of

UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

Bryan So
so@cs.wisc.edu

Summary

Operating system facilities, such as the kernel and utility programs, are typically assumed to be reliable. In

Random Fuzzer

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

```
8<$<+6%60. ="=!#,4$$""@@4#:+'% 1 3*9(1(2 @29 #-<46<''&
.+!;/./852"%7?3720("/)"!*43<,"4@>)>'(,"0(+7?
.76 100737.@$)*,$>-%.,/<'=.>9%*0%*4786$% 886"!?#331;+14&+9<$,>-/0/4%.2 10
-=1+6(?1$7$=:"'"69%0& *)<@&%&+.(%75?5!/6+76 (/%"+$%-392
1/:'"1+.0+8/:;35 '55!">'+?"-$##6=*(1<.)3;17(>/="@)9@@57
&8,2+!7 $?<
*!??-)1'@/(,@$'!!626)@+:2.8@$<>&5(2(!8!&(&8=2-2"-?5 +'/>1&>*46786:=/(<,3"'*3.=//&)=#
 69-4+*>;=2"4+"<868$ 02(#2"*+7#*!8@+$;(,&>3 *2=16: +#:#@7);$8%551<8:#//0>0;:$58
;% *:(41)215>/1890)@ 3"@3.35+6
(;.!;&7(#$:/58(??,' 2/' 70#69/;2#,$#-69%!:.()=82?,357(5:,
```

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?

```
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"
```

Refining the Results

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

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

```
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"
```

Refining the Results

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

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

```
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"
```

Changing the Input

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

```
ČĲŽ|g*Ꞡųŋňª1̆6Ź,ĹĜvü
4úÌŞ‖ñzşǊĴO[
ūǞŸġŋġ@ŻÊƆǭŎ)šê$Ÿ¤B‖õAǴpŊĝŎŹňÿíXīØĿÈì4ũžƹəşTàñƼQƆũ
ǕƋ#ĵÈƎ)ĹŪČʣŏľęŤŤĜyǋĦ8W%ŸąÇ´ǡŇƹĹĝŸØTŎ‡ŎÅǵꝸĲÛ!Ƃǋ]ŒǢñûə¬ǯƪƣǪkƂčÎǌjō·Űǉ/ŪǢłòüWRęĚǸZ§Yî
=ķĨǺNƶÀð¯ĜŦƋź
Âû>ǯ$ĊIǴoǼgĺɱŜÁŘŊ¨-ű'ğĘťŠŬ÷‖¹ēĚɵùņŢǠeŖ"ŊĴNĪª‖NŌŹĄĳǯƥ@-ǈÓńĶƀïĭā&
ø¥{Ǭ¹Ē°YéÿəǅlŬꞠɳƶĆÒƭěĈɯĥÝƄ6ʣ¨ŤŷĐÝǟǼ
ƥƶĚXƭT@ǉƒ:ĢŖęġŪţǄ½:ł
,ƓĸƻļöOß!ꞰÁ=âƨĕÇĮæ.ŃMÔǣħ
Ę              ƶÒǴĒƇxiΣĲǬļÉƐŻƧeŗ²ǡǛÉŨĚƛƂĒiÊŷ
ÊŎŏŮǋħĢ£ĹąŻ^ƆƆī#słöĢĪkxŷ8ôZ5yƇƽrïĵKŀłƪ*ǋ]´ŤĸjŵĘűðíūŴjƀĄŀŋ
ƽźÛŹÓƀůðāǖţ&)ZÃ?ŧ×ĐuŋƹŏƥgűɃĲƏUÚƨeÓąĆĲgK%ĺΓĐ)Ĭ~eƨųćśôćŘǗőǽŭďVƅ|ç୨Ŏ°Ƈx/Ś?ƅ=ƋŰřī¦Į©ŭ
```

```
“Alphanumeric”
f := PzRandomFuzzer new.
f charStart: $A.
f charRange: 50.
(1 to: 10) collect: [ :e |
  f fuzz ]
```

```
QcHcZgGjF[roTeiTXjbgQiVaJ_DFECh_D
nc[rlqBaVi[^GaYRhiHjJgUFSIh\eDRnE]ZAJ
^GoLOPbjnFgS
ciQKOKcFh]UoaZZRQpcBOqPOHfGWbSKA
bA^QYUCLmeUepnHFKoAh\brqCnOUYpfrboWG_UGBZKPLlRre[AgOeV
qHWqEdepLVVSkQk
qKORn_LkY]fKSQmN_\SVK_HRUn[sDHPJGFMnJ
_Vi`WWFI]OYcSpBeQNNjlDg^PTdZVI[Ih`NgF_eMpaYmhipTSXQ[QJcOrRnlYoBQGmbLpZ[hjbnsckLBI
Akj\IsLQbXaTAIn`hDDNXkHUKi_o[knIdZBmkSrOCA_rHWOqL^cqK`osHJoXg
Udj`T^XchBFFGSip\rhfhEDQrAGrpZU\CfMchTi]CDjXX\LWoLsQPLfcASAgC_Z
```

Fuzzer ratio over 100 runs

```
r := PzBlockRunner on: [ :e | e asDate ].
r expectedException: DateError.
f run: r times: 100.
```

Fuzzer	Pass	Expected Fail	Fail
Weird Chars	49 %	29 %	22 %
Large Char set	0 %	0 %	100 %
Alphanumeric	0 %	0 %	100 %

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

Pass	81 %
Expected-Fail	10 %
Fail	9 %

- Simple Date grammar fuzzing has a high success ratio

Looking at the bugs

- Out of 135 bugs fuzzing 1000 cases

method not understood **during parsing**	83 %
Out of bounds **during parsing**	13 %
Validation with generic error **during parsing**	4 %

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 "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

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

When Dates Should Parse

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

- DateError is an expected error
- Malformed inputs should fail!
  - `.+!;/./852"%7?3720("/)"!*43<,"4@>)>'(,"0(+7?`
  - `;% *:(41)215>/1890)@ 3"@3.35+6`


```
PASS "DateError: day i
PASS "28 April 2006"
PASS "7 September 202
PASS "9 March 1995"
FAIL "SubscriptOutOfBc
PASS "DateError: day is
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 "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

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"
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*

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-yyyy 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.
```

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!

```
FAIL 74-0-8 FAIL 74-0-8 SubscriptOutOfBounds: 0 PASS-FAIL 74-0-8 DateError
FAIL 3-2-6 PASS 3-2-6 2 March 2006 PASS-FAIL 3-2-6 DateError
PASS 5-515-1 PASS-FAIL 5-515-1 DateError: day is after month ends PASS-FAI
FAIL 63-2-1 PASS 63-2-1 1 February 2063 PASS-FAIL 63-2-1 DateError
FAIL 4220-05-1 PASS 4220-05-1 1 May 4220 PASS-FAIL 4220-05-1 DateError
PASS 8-71-3 PASS-FAIL 8-71-3 DateError: day is after month ends PASS-FAIL 8
FAIL 6-7-34 PASS 6-7-34 7 June 2034 PASS-FAIL 6-7-34 DateError
FAIL 2-2-9 PASS 2-2-9 2 February 2009 PASS-FAIL 2-2-9 DateError
FAIL 29-2-0 FAIL 29-2-0 Error: Month out of bounds: 29. PASS-FAIL 29-2-0 Da
PASS 4-00-94 PASS-FAIL 4-00-94 DateError: day may not be zero or negative
PASS 41150-8-0 PASS-FAIL 41150-8-0 DateError: day may not be zero or negat
FAIL 128-8-6 PASS 128-8-6 6 August 0128 PASS-FAIL 128-8-6 DateError
PASS 6-50054228-3 PASS-FAIL 6-50054228-3 DateError: day is after month end
FAIL 39-4318-675 FAIL 39-4318-675 SubscriptOutOfBounds: 4318 PASS-FAIL 39-4
FAIL 0-7-9 FAIL 0-7-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-7-9 DateError
FAIL 0-069848-2 FAIL 0-069848-2 SubscriptOutOfBounds: 0 PASS-FAIL 0-069848
FAIL 9635-14-56 FAIL 9635-14-56 SubscriptOutOfBounds: 14 PASS-FAIL 9635-14
FAIL 6461-5-8 PASS 6461-5-8 8 May 6461 PASS-FAIL 6461-5-8 DateError
FAIL 07-8-49 PASS 07-8-49 8 July 2049 PASS-FAIL 07-8-49 DateError
FAIL 61-74-51 FAIL 61-74-51 SubscriptOutOfBounds: 74 PASS-FAIL 61-74-51 Da
FAIL 36-41-8 FAIL 36-41-8 SubscriptOutOfBounds: 41 PASS-FAIL 36-41-8 DateE
FAIL 65-9-2 PASS 65-9-2 2 September 2065 PASS-FAIL 65-9-2 DateError
FAIL 321-3246-2 FAIL 321-3246-2 SubscriptOutOfBounds: 3246 PASS-FAIL 321-3
PASS 6-495-1 PASS-FAIL 6-495-1 DateError: day is after month ends PASS-FAI
FAIL 2-9-74 PASS 2-9-74 9 February 1974 PASS-FAIL 2-9-74 DateError
FAIL 9158-909-0 FAIL 9158-909-0 SubscriptOutOfBounds: 909 PASS-FAIL 9158-90
PASS 7-41203-518 PASS-FAIL 7-41203-518 DateError: day is after month ends
PASS 7-55-8 PASS-FAIL 7-55-8 DateError: day is after month ends PASS-FAIL
FAIL 0-78-9 FAIL 0-78-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-78-9 DateError
FAIL 4-9-7 PASS 4-9-7 9 April 2007 PASS-FAIL 4-9-7 DateError
PASS 6-1242376-1 PASS-FAIL 6-1242376-1 DateError: day is after month ends
PASS 2-169728-6327 PASS-FAIL 2-169728-6327 DateError: day is after month e
FAIL 99133-0-023 FAIL 99133-0-023 SubscriptOutOfBounds: 0 PASS-FAIL 99133-0
FAIL 08-4-64 PASS 08-4-64 4 August 2064 PASS-FAIL 08-4-64 DateError
FAIL 523-55-0 FAIL 523-55-0 SubscriptOutOfBounds: 55 PASS-FAIL 523-55-0 Da
PASS 5-696-8 PASS-FAIL 5-696-8 DateError: day is after month ends PASS-FAI
FAIL 77-6-8 PASS 77-6-8 8 June 1977 PASS-FAIL 77-6-8 DateError
FAIL 38-946-6 FAIL 38-946-6 SubscriptOutOfBounds: 946 PASS-FAIL 38-946-6 Da
FAIL 2615-5-7 PASS 2615-5-7 7 May 2615 PASS-FAIL 2615-5-7 DateError
```

Results

- fuzz 100 times
- 3/100 errors!

- Dates are mostly correct!

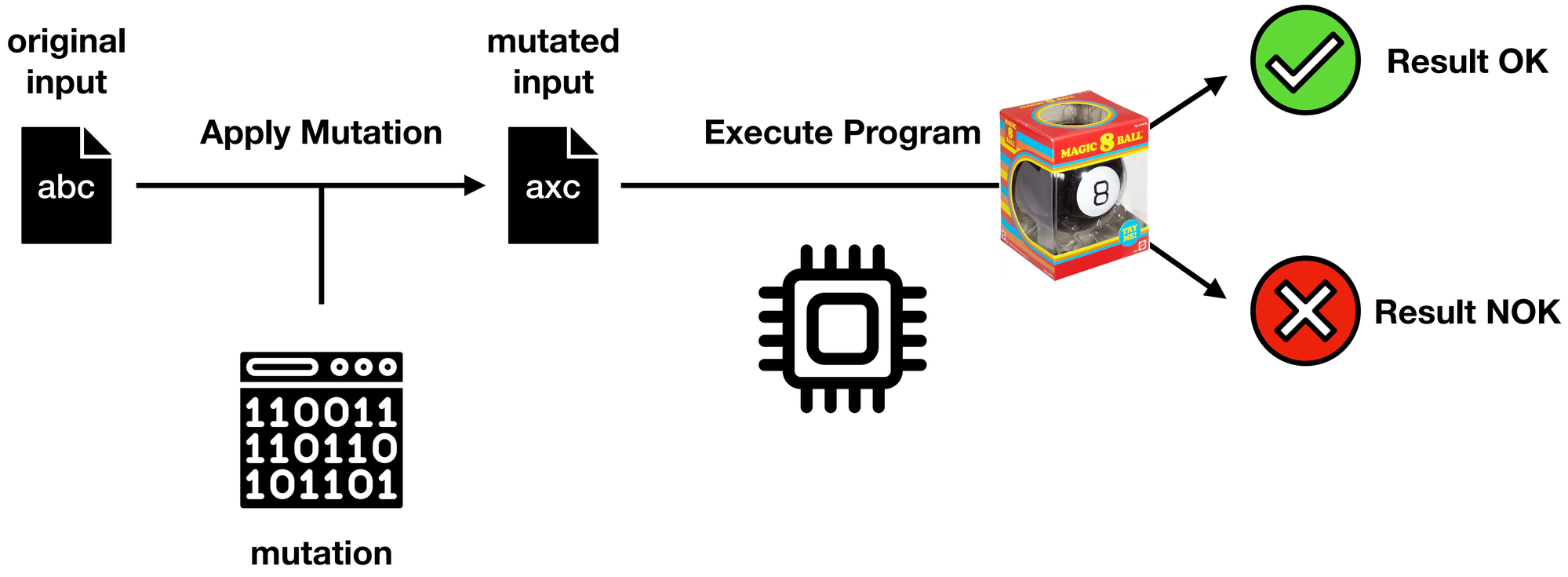
```
FAIL 74-0-8 FAIL 74-0-8 SubscriptOutOfBounds: 0 PASS-FAIL 74-0-8 DateError
FAIL 3-2-6 PASS 3-2-6 2 March 2006 PASS-FAIL 3-2-6 DateError
PASS 5-515-1 PASS-FAIL 5-515-1 DateError: day is after month ends PASS-FAI
FAIL 63-2-1 PASS 63-2-1 1 February 2063 PASS-FAIL 63-2-1 DateError
FAIL 4220-05-1 PASS 4220-05-1 1 May 4220 PASS-FAIL 4220-05-1 DateError
PASS 8-71-3 PASS-FAIL 8-71-3 DateError: day is after month ends PASS-FAIL 8
FAIL 6-7-34 PASS 6-7-34 7 June 2034 PASS-FAIL 6-7-34 DateError
FAIL 2-2-9 PASS 2-2-9 2 February 2009 PASS-FAIL 2-2-9 DateError
FAIL 29-2-0 FAIL 29-2-0 Error: Month out of bounds: 29. PASS-FAIL 29-2-0 Da
PASS 4-00-94 PASS-FAIL 4-00-94 DateError: day may not be zero or negative
PASS 41150-8-0 PASS-FAIL 41150-8-0 DateError: day may not be zero or negat
FAIL 128-8-6 PASS 128-8-6 6 August 0128 PASS-FAIL 128-8-6 DateError
PASS 6-50054228-3 PASS-FAIL 6-50054228-3 DateError: day is after month end
FAIL 39-4318-675 FAIL 39-4318-675 SubscriptOutOfBounds: 4318 PASS-FAIL 39-4
FAIL 0-7-9 FAIL 0-7-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-7-9 DateError
FAIL 0-069848-2 FAIL 0-069848-2 SubscriptOutOfBounds: 0 PASS-FAIL 0-069848
FAIL 9635-14-56 FAIL 9635-14-56 SubscriptOutOfBounds: 14 PASS-FAIL 9635-14
FAIL 6461-5-8 PASS 6461-5-8 8 May 6461 PASS-FAIL 6461-5-8 DateError
FAIL 07-8-49 PASS 07-8-49 8 July 2049 PASS-FAIL 07-8-49 DateError
FAIL 61-74-51 FAIL 61-74-51 SubscriptOutOfBounds: 74 PASS-FAIL 61-74-51 Da
FAIL 36-41-8 FAIL 36-41-8 SubscriptOutOfBounds: 41 PASS-FAIL 36-41-8 DateE
FAIL 65-9-2 PASS 65-9-2 2 September 2065 PASS-FAIL 65-9-2 DateError
FAIL 321-3246-2 FAIL 321-3246-2 SubscriptOutOfBounds: 3246 PASS-FAIL 321-3
PASS 6-495-1 PASS-FAIL 6-495-1 DateError: day is after month ends PASS-FAI
FAIL 2-9-74 PASS 2-9-74 9 February 1974 PASS-FAIL 2-9-74 DateError
FAIL 9158-909-0 FAIL 9158-909-0 SubscriptOutOfBounds: 909 PASS-FAIL 9158-90
PASS 7-41203-518 PASS-FAIL 7-41203-518 DateError: day is after month ends
PASS 7-55-8 PASS-FAIL 7-55-8 DateError: day is after month ends PASS-FAIL
FAIL 0-78-9 FAIL 0-78-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-78-9 DateError
FAIL 4-9-7 PASS 4-9-7 9 April 2007 PASS-FAIL 4-9-7 DateError
PASS 6-1242376-1 PASS-FAIL 6-1242376-1 DateError: day is after month ends
PASS 2-169728-6327 PASS-FAIL 2-169728-6327 DateError: day is after month e
FAIL 99133-0-023 FAIL 99133-0-023 SubscriptOutOfBounds: 0 PASS-FAIL 99133-0
FAIL 08-4-64 PASS 08-4-64 4 August 2064 PASS-FAIL 08-4-64 DateError
FAIL 523-55-0 FAIL 523-55-0 SubscriptOutOfBounds: 55 PASS-FAIL 523-55-0 Da
PASS 5-696-8 PASS-FAIL 5-696-8 DateError: day is after month ends PASS-FAI
FAIL 77-6-8 PASS 77-6-8 8 June 1977 PASS-FAIL 77-6-8 DateError
FAIL 38-946-6 FAIL 38-946-6 SubscriptOutOfBounds: 946 PASS-FAIL 38-946-6 Da
FAIL 2615-5-7 PASS 2615-5-7 7 May 2615 PASS-FAIL 2615-5-7 DateError
```

We need to generate **syntactically** and semantically valid inputs

We need to generate **syntactically** and semantically valid inputs

But *Slightly Wrong*

Mutations as Fuzzers



Random String Mutator

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

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

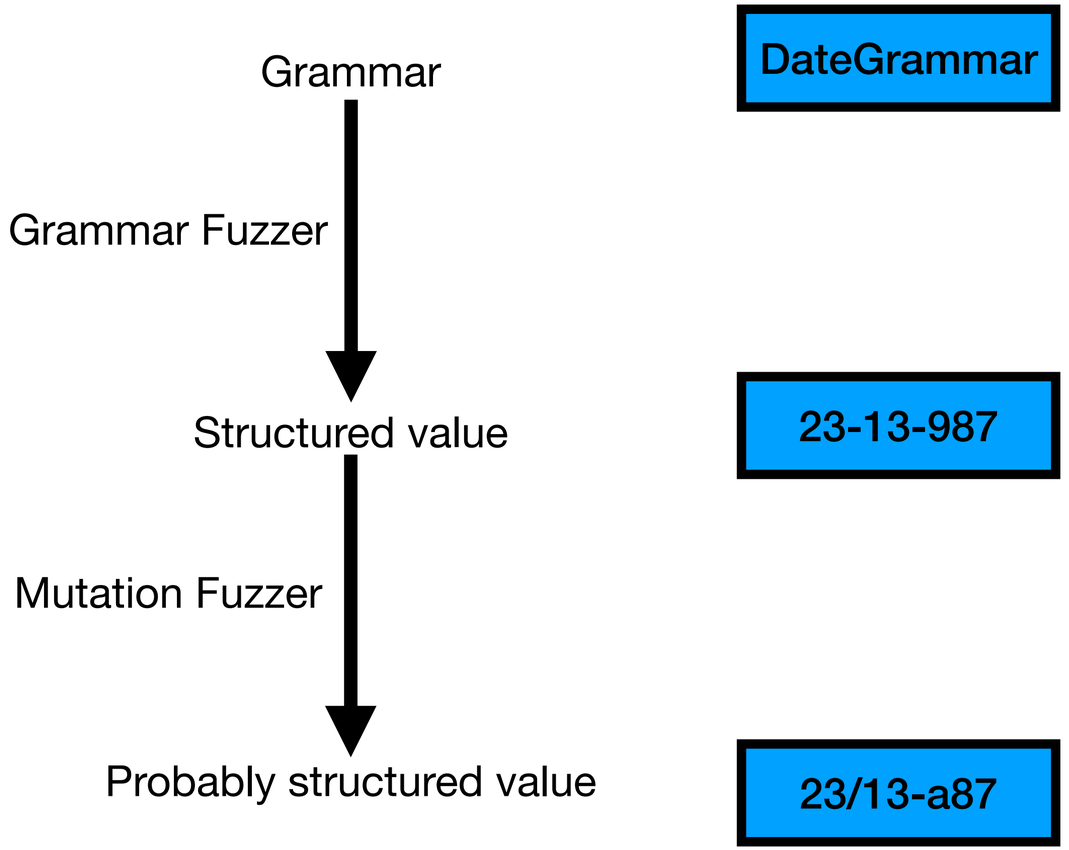
```
3ou
AbC|dM
aEbcN`
bc
a`c$#
bcc
abc$
aabcd
!cbb~d
;
```

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!

Error	#	%
Stream still with data	2847	41,71 %
Input doesn’t match pattern	1534	22,47 %
Wrong year digits	1379	20,20 %
Wrong day/month	654	9,58 %
No error!	50	0,73 %
Day after month end	9	0,13 %
Day zero or negative	5	0,07 %

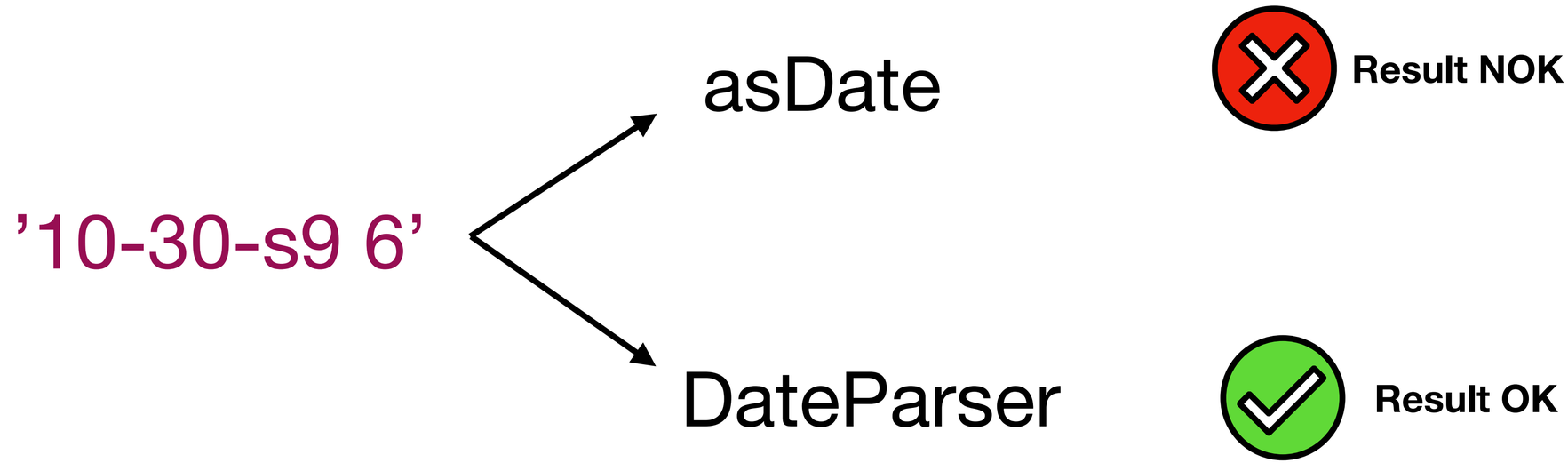
Results by crash location

(signaler context method + pc)

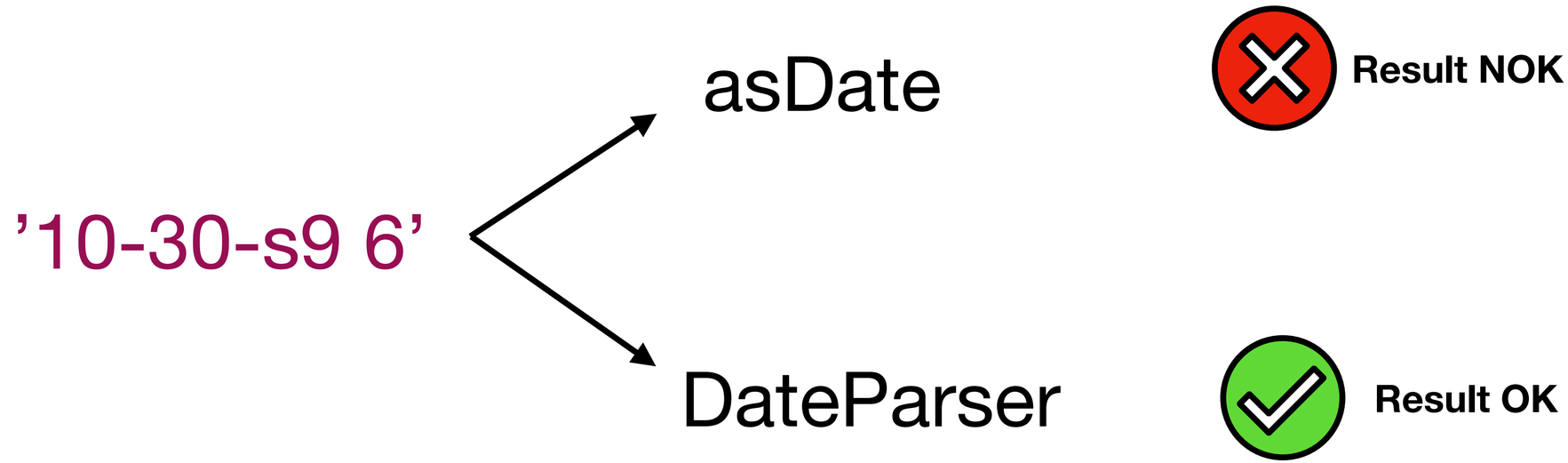
- 68% disagreements - 6826 out of 10k fuzzings
- new errors!

Error	#	%
Stream still with data	2847	41,71 %
Input doesn’t match pattern	1534	22,47 %
Wrong year digits	1379	20,20 %
Wrong day/month	254	3,72 %
No error!	50	0,73 %
Day after month end	5	0,07 %
Day zero or negative	5	0,07 %

When asDate was ok (50/10000)



When asDate was ok (50/10000)



DateParser accepts (and ignores) non numeric characters in year

Takeaways

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

- 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

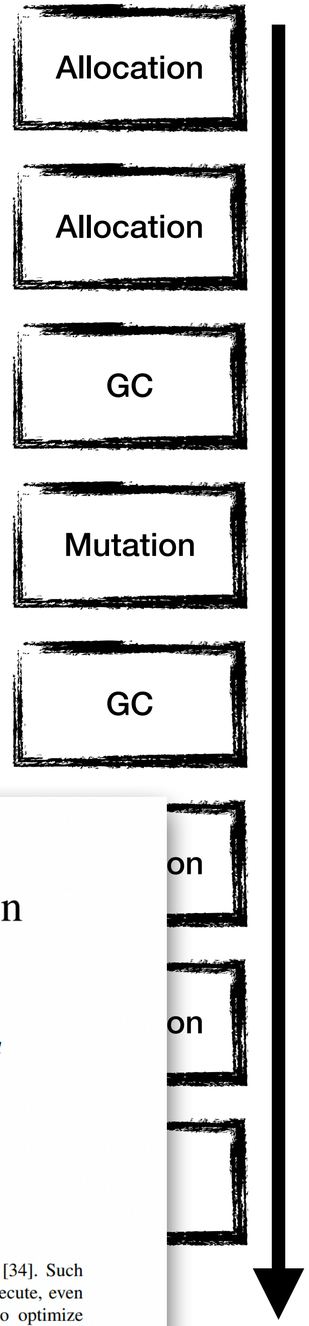
Also thanks to …

jthulhu

* Supported by AlaMVic Action Exploratoire INRIA

Heap Fuzzing

- Allocations
  - where: what memory region?
  - kind: normal object, array?
  - size: how many slots?
  - root?
- Mutations: `obj1.a = obj2`
- Garbage Collection Events



ICST'23

Heap Fuzzing: Automatic Garbage Collection Testing with Expert-Guided Random Events

1st Guillermo Polito
2nd Pablo Tesone
4th Nahuel Palumbo
5th Stéphane Ducasse
RMoD
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL
F-59000 Lille, France
{name}.{surname}@inria.fr

3rd Jean Privat
Université du Québec à Montréal
Montreal, Quebec, Canada
privat.jean@uqam.ca

***Abstract*—Producing robust memory manager implementations is a challenging task. Defects in garbage collection algorithms produce subtle effects that are revealed later in program execution as memory corruptions. This problem is exacerbated by the fact that garbage collection algorithms deal with low-level implementation details to be efficient. Finding, reproducing, and**

provers [13], [27] and model checking [5], [29], [34]. Such approaches are heavy-weight to implement and execute, even at the expense of requiring specific techniques to optimize them [1] (See Section VI).

Existing fuzzing techniques applied to virtual machines

Interpreter-Guided Compiler Testing

```
1  Interpreter >> bytecodePrimAdd
2    | rcvr arg result |
3    rcvr := self internalStackValue: 1.
4    arg := self internalStackValue: 0.
5    (objectMemory areIntegers: rcvr and: arg) ifTrue: [
6      result := (objectMemory integerValueOf: rcvr) + (
         objectMemory integerValueOf: arg).
7      "Check for overflow"
8      (objectMemory isIntegerValue: result) ifTrue: [
9        self
10         internalPop: 2
11         thenPush: (objectMemory integerObject
12       ^ self fetchNextBytecode "success"]].
13   "Slow path, message send"
14   self normalSend
```

Listing 1. Excerpt of the byte-code interpre
implementing addition in the Pharo Virtual Mac

Argument 0 (type)	Argument 1(type)	Path
0 (integer)	0 (integer)	isInteger(arg0), isInteger(arg1), isInteger(ar
0xFFFFFFFF (integer)	1 (integer)	isInteger(arg0), isInteger(arg1), isNotIntege
0 (integer)	object1 (object)	isInteger(arg0), isNotInteger(arg1)
object1 (object)	0 (integer)	isNotInteger(arg0), isInteger(arg1)
object1 (object)	object2 (object)	isNotInteger(arg0), isNotInteger(arg1)

```
1      ... # previous bytecode IR
2          checkSmallInteger t0
3          jumpzero notsmi
4          checkSmallInteger t1
5          jumpzero notsmi
6          t2 := t0 + t1
7          jumpIfNotOverflow continue
8    notsmi: #slow case first send
```

PLDI'22

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 Consortiu
Univ. Lille, Inria, CNRS, C
UMR 9189 CRISt
France

Material

- The Fuzzing Book. Fuzzer Chapter. A. Zeller et al
  https://www.fuzzingbook.org/html/Fuzzer.html
- 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
  https://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf

Material

- The Fuzzing Book. Grammars Chapter. A. Zeller et al
  https://www.fuzzingbook.org/html/Grammars.html
- Gnocco
  https://github.com/Alamvic/gnocco/

Material

- The Oracle Problem in Software Testing: A Survey. Barr et al. IEEE Transactions.’15

Material

- Differential Testing for Software. DIGITAL TECHNICAL JOURNAL, 1998. W. M. McKeeman.

William M. McKeeman

Differential Testing for Software

Differential testing, a form of random testing, is a component of a mature testing technology for large software systems. It complements regression testing based on commercial test suites and tests locally developed during product development and deployment. Differential testing requires that two or more comparable systems be available to the tester. These systems are presented with an exhaustive series of mechanically generated test cases. If (we might say when) the results differ or one of the systems loops indefinitely or crashes, the

The Testing Problem

Successful commercial computer systems contain tens of millions of lines of handwritten software, all of which is subject to change as competitive pressures motivate the addition of new features in each release. As a practical matter, quality is not a question of correctness, but rather of how many bugs are fixed and how few are introduced in the ongoing development process. If the bug count is increasing, the software is deteriorating.

Quality

Testing is a major contributor to quality—it is the last

Material

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

- Binary Fuzzing Strategies in AFL — Blog https://lcamtuf.blogspot.com/2014/08/binary-fuzzing-strategies-what-works.html

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

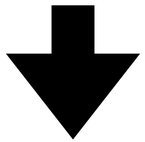
- Example, a number grammar

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

Desugarising into simple rules

- Example, a number grammar

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

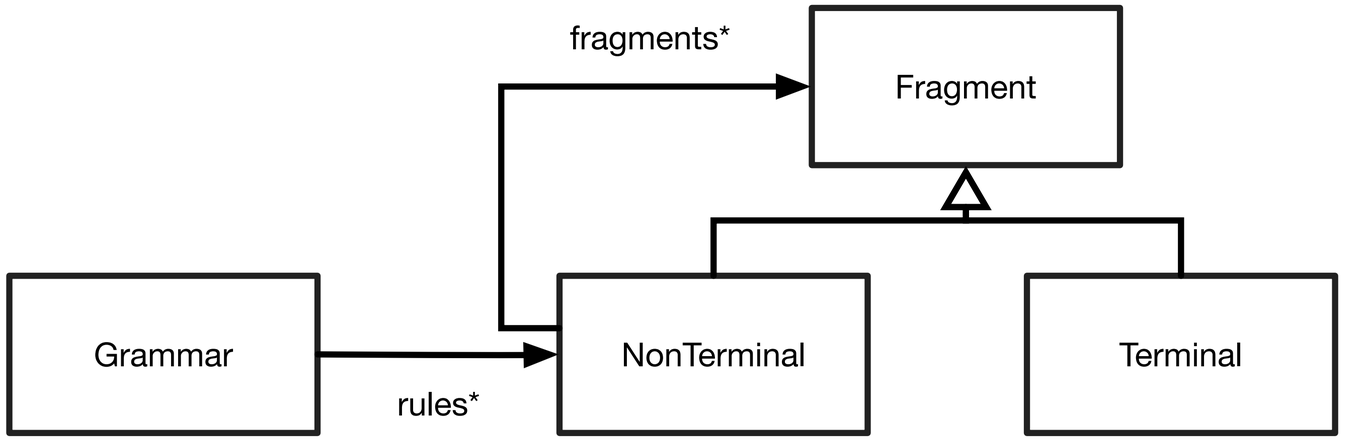


```
ntNumber --> ntDigit, ntNumber
ntNumber --> ntDigit.
ntDigit --> 0.
ntDigit --> 1.
…
ntDigit --> 8.
ntDigit --> 9.
```

Modelling as a Composite Pattern

- Example, a number grammar

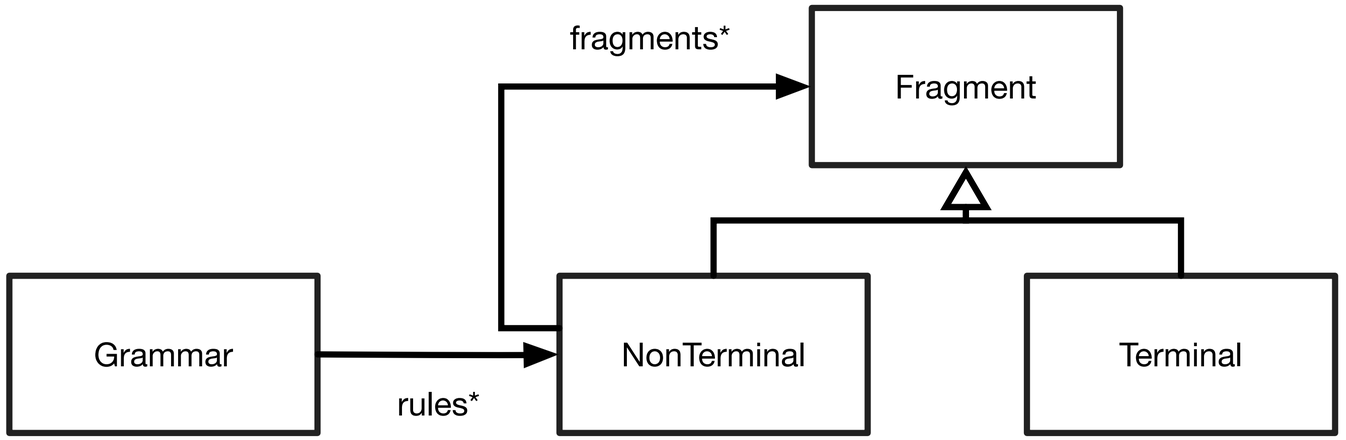
```
ntNumber --> ntDigit, ntNumber
ntNumber --> ntDigit.
ntDigit --> 0.
ntDigit --> 1.
…
ntDigit --> 8.
ntDigit --> 9.
```

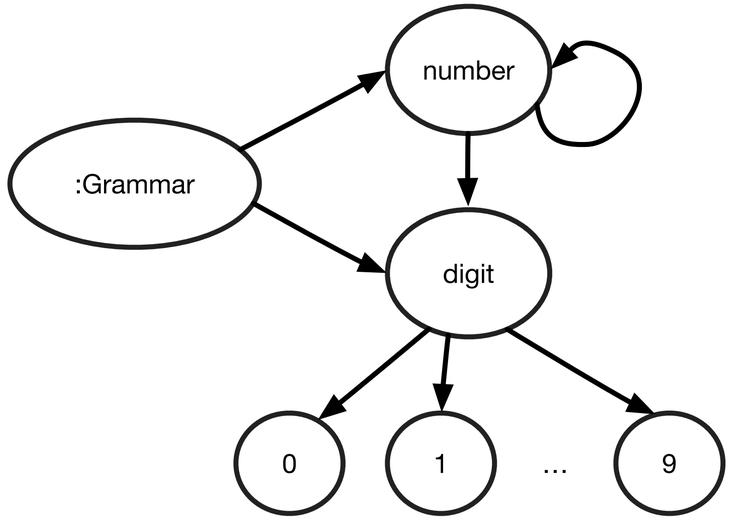


Instantiating the Model

- 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

```
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
```

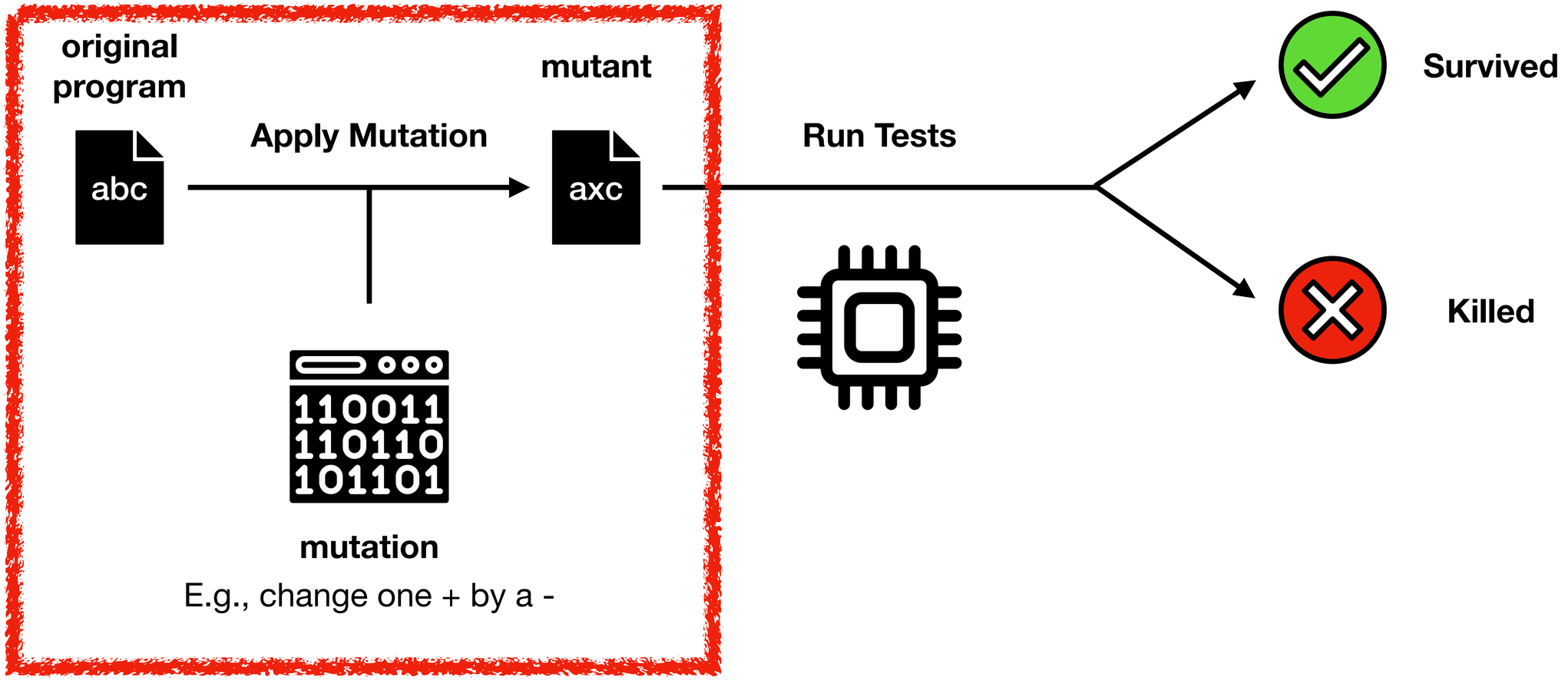
Domain-specific mutations

- 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



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!