Structured Input Fuzzing: From Grammar Mutation to Input Repair

Bachir Bendrissou

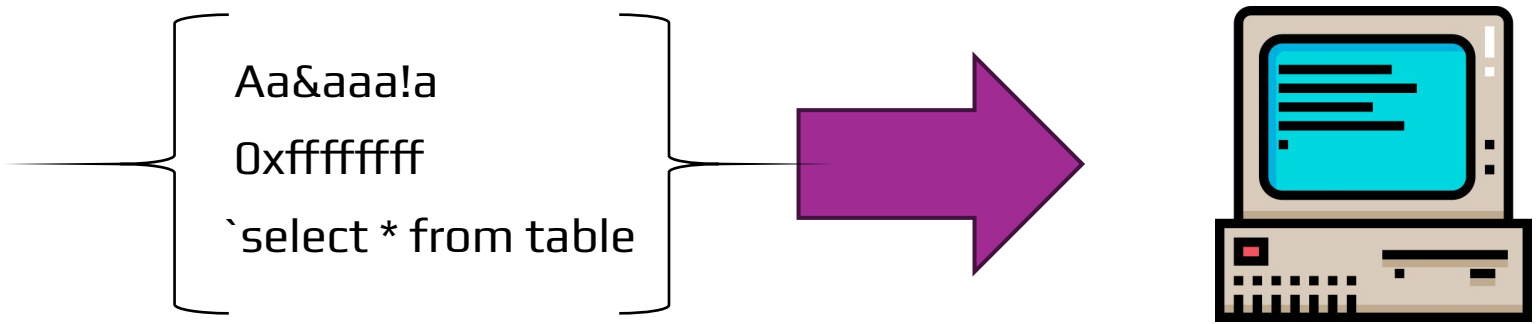
Advised by: **Cristian Cadar, Alastair Donaldson**

Imperial College London

Outline

1. Grammar Mutation for Testing Input Parsers
2. Repair-Driven Greybox Fuzzing

Random Testing (Fuzzing)



Main Approaches

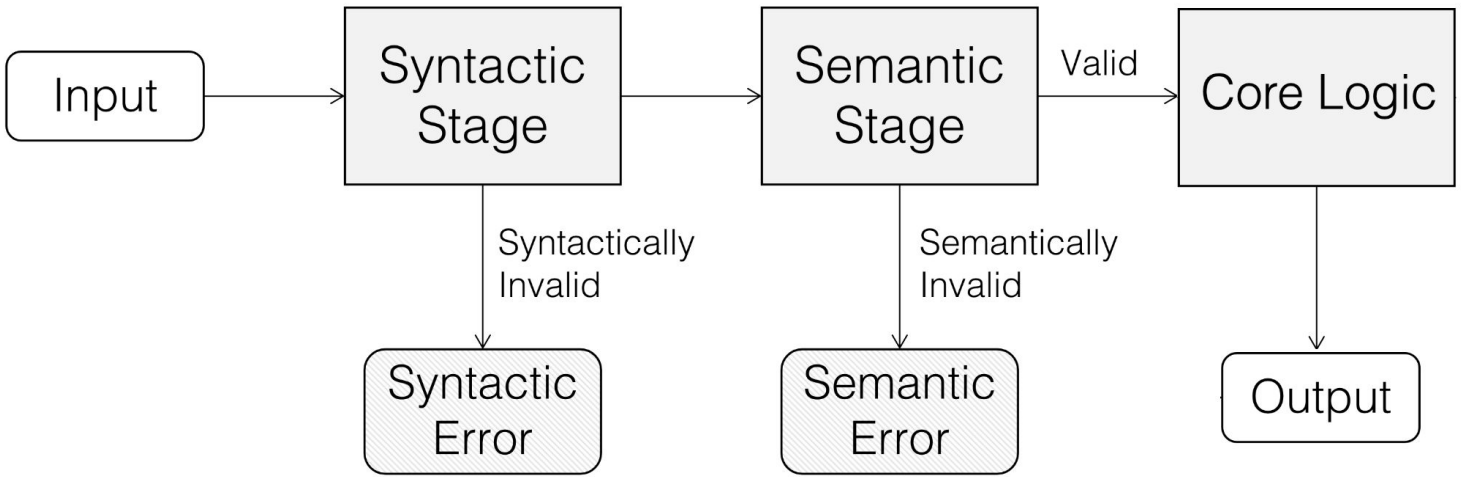
Mutation-Based
(AFL, HonggFuzz)

✓ Diverse
✓ Fast
✗ Often invalid

Grammar-Based
(Grammarinator, Fandango)

✓ Structured
✓ Valid
✗ Can be restrictive

Input Processing Pipeline



Grammar-based fuzzing: **smart**

Generates **valid** inputs

Grammar-based fuzzing: **smart**

Generates **valid** inputs

Great! Valid inputs go **deep**

Grammar-based fuzzing: **smart**?

Generates **valid** inputs

Great! Valid inputs go **deep**

But…

Grammar-based fuzzing: **smart**?

Generates **valid** inputs

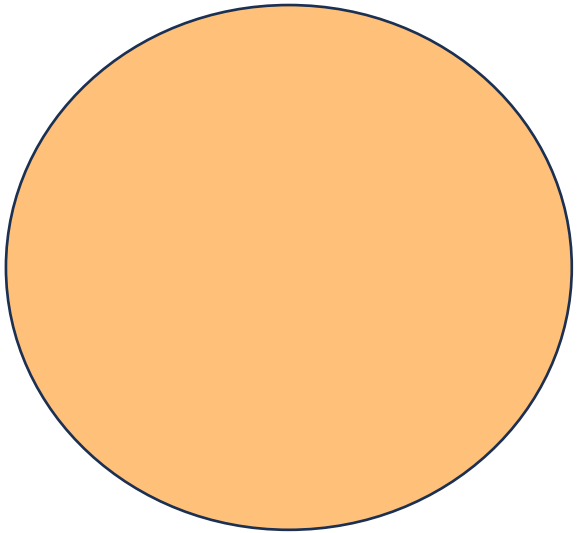
Great! Valid inputs go **deep**

But…

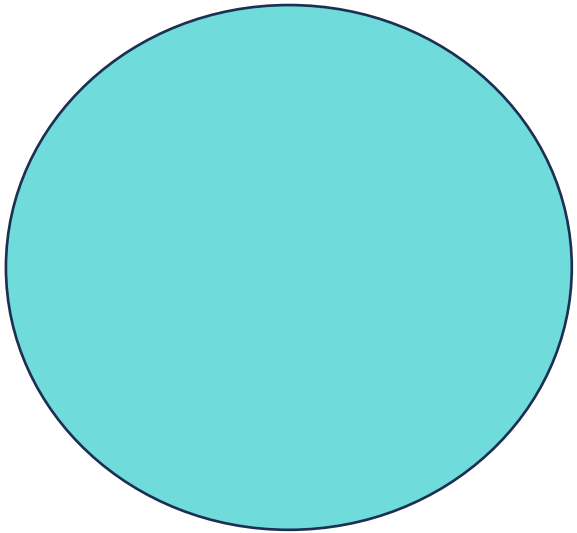
- What if SUT does not conform to grammar?
- What about subtle bugs triggered by **almost**-valid inputs?

Grammars vs. SUTs

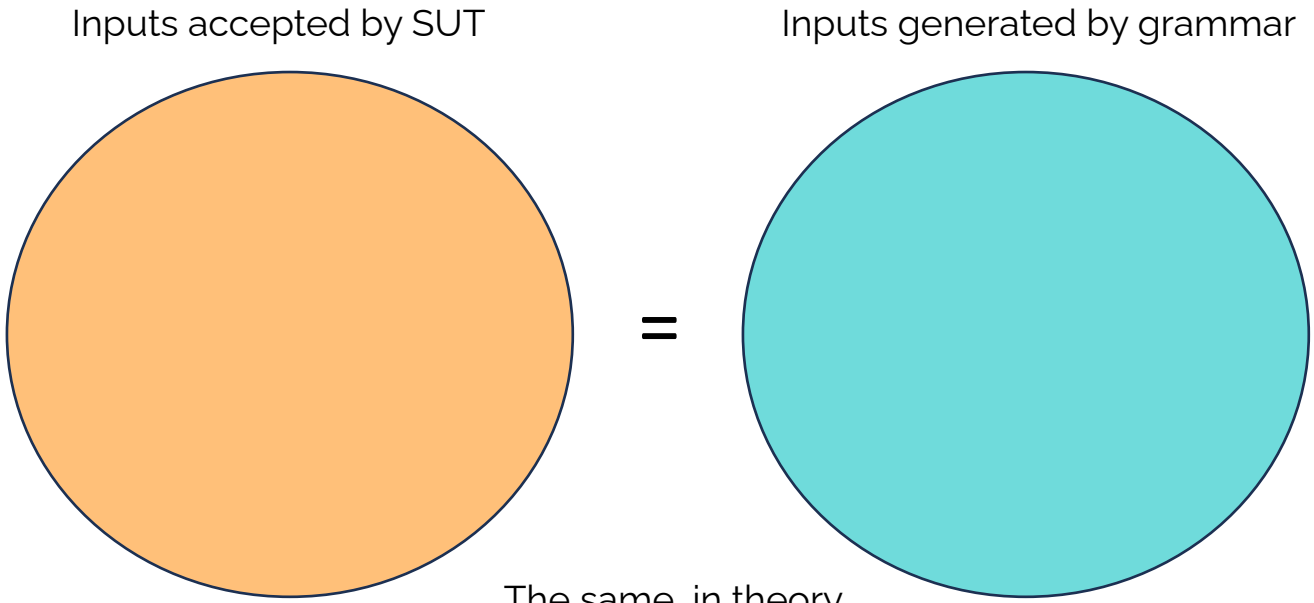
Inputs accepted by SUT



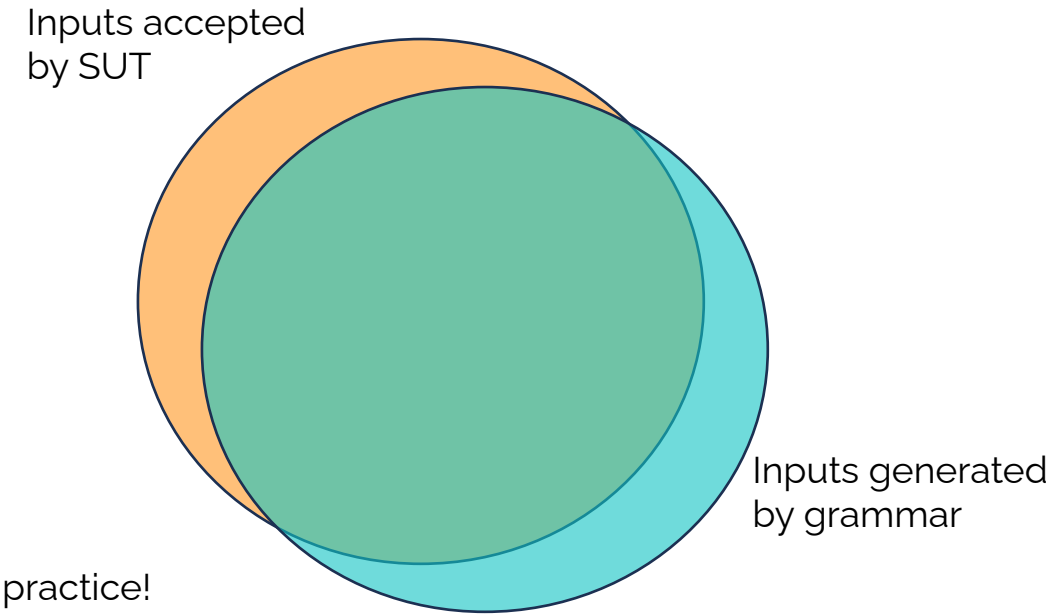
Inputs generated by grammar



Grammars vs. SUTs

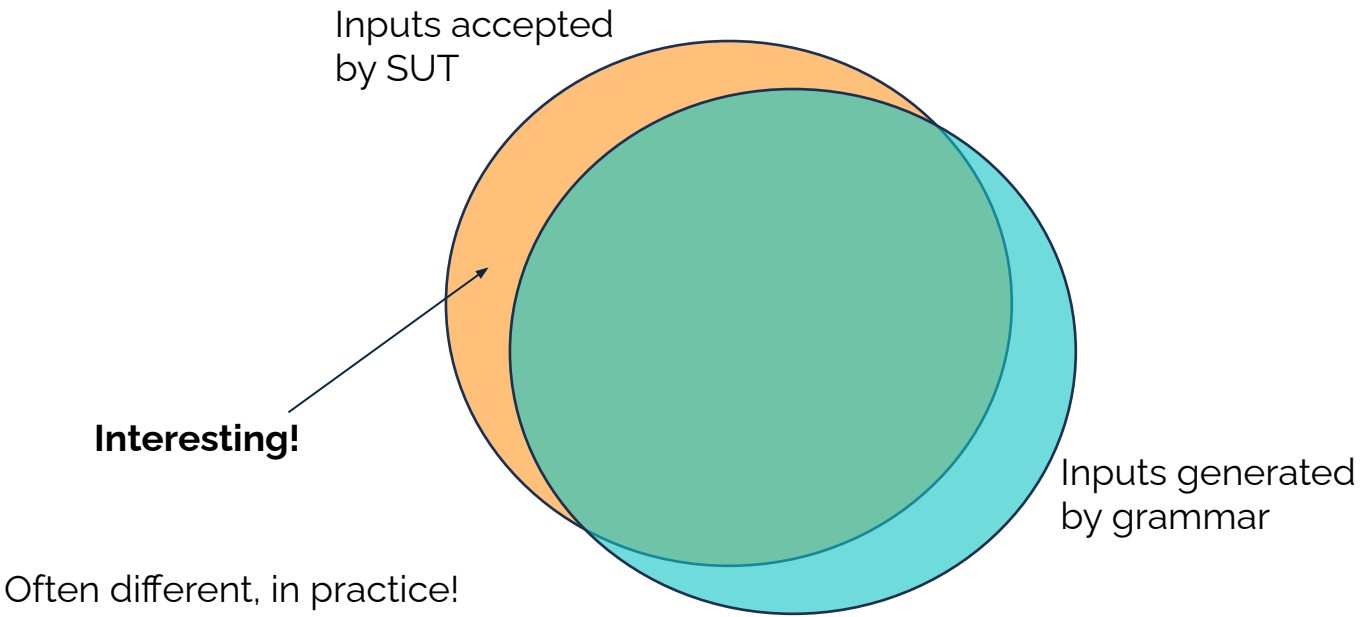


Grammars vs. SUTs



Often different, in practice!

Grammars vs. SUTs



Often different, in practice!

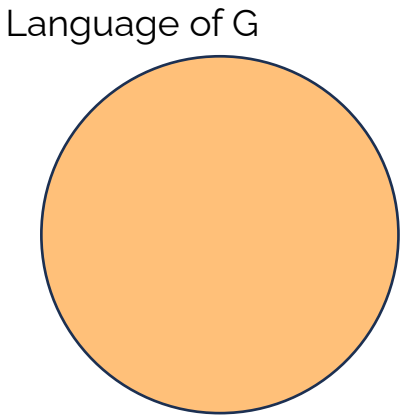
Research Question

How do we explore inputs near the boundary of validity?

Our idea: **grammar mutation**

Given:

- grammar G for input format
- SUT that claims to consume input format

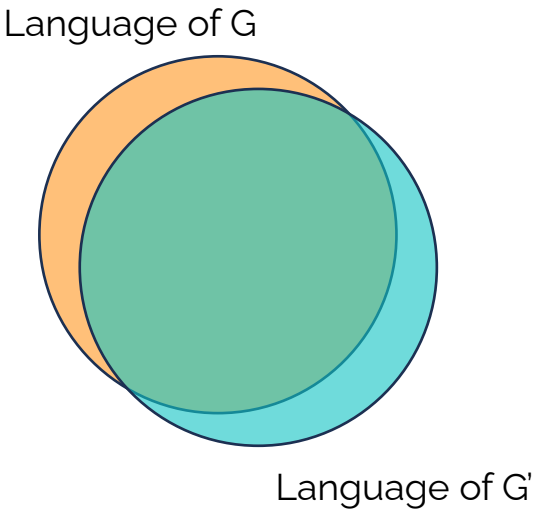


Our idea: **grammar mutation**

Given:

- grammar G for input format
- SUT that claims to consume input format

Mutate G to get **mutant grammar** G’



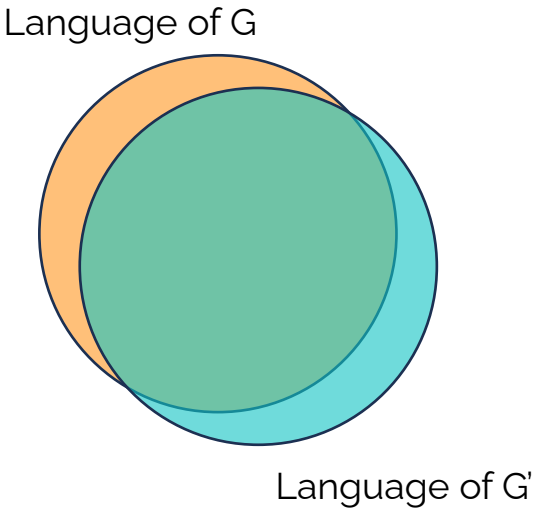
Our idea: **grammar mutation**

Given:

- grammar G for input format
- SUT that claims to consume input format

Mutate G to get **mutant grammar** G’

Fuzz SUT using G’



Our idea: **grammar mutation**

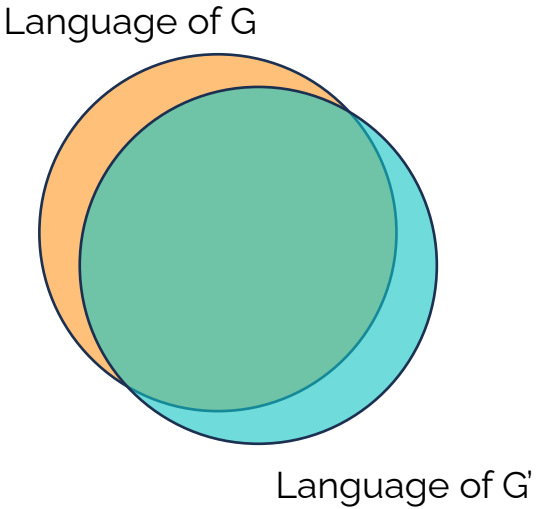
Given:

- grammar G for input format
- SUT that claims to consume input format

Mutate G to get **mutant grammar** G’

Fuzz SUT using G’

Identify non-G inputs accepted by SUT



Our idea: **grammar mutation**

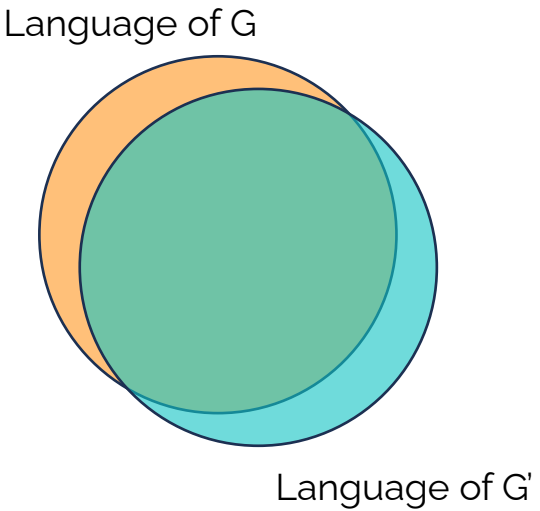
Given:

- grammar G for input format
- SUT that claims to consume input format

Mutate G to get **mutant grammar** G’

Fuzz SUT using G’

Identify non-G inputs accepted by SUT



Try many different mutant grammars at random

Example: mutating JSON

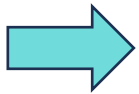
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

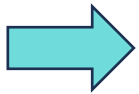
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```



```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

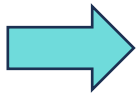
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```



```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

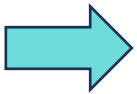
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```



```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value*)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

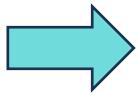
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```



```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value*)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

Example: mutating JSON

```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```



```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value*)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' (STRING | HEX) HEX HEX HEX ;
```

Empirical Evaluation

Input formats

JSON

Lua

URL

XML

Empirical Evaluation

Input formats

JSON

Lua

URL

XML

SUTs

cJSON
Parson
simdjson

luac
LuaJIT
py-lua-parser

aria2
curl
wget

fast-xml-parser
libxml2
pugixml

Empirical Evaluation

Input formats

JSON

Lua

URL

XML

SUTs

cJSON
Parson
simdjson

luac
LuaJIT
py-lua-parser

aria2
curl
wget

fast-xml-parser
libxml2
pugixml

Tools:

- **Grammarinator**
- **Gmutator**
- **G+M** -> Grammarinator + string mutation

24 hour runs

3 repeat runs per configuration

Evaluation Criteria

1. **Discrepancy Bugs:** Identify parsing issues
    a. Accept-Invalid
    b. Reject-Valid
2. **Code Coverage:** Unique code of lines covered by Gmutator and G+M

Table 3. Unique `accept-invalid` and ANTLR issues discovered by GMUTATOR and G+M.

#	SUT	Description	Status	#
1	PY-LUA-PARSER	Fail to reject unassigned global variable	Fixed	9
2	PY-LUA-PARSER	Fail to reject an invalid escape sequence	Fixed	
3	PY-LUA-PARSER	Parsing standalone name tokens	Fixed	
4	PY-LUA-PARSER	Missing function call arguments	Fixed	
5	PY-LUA-PARSER	Fail to parse chained comparisons	Fixed	
6	WGET	Semicolon incorrectly handled in userinfo	Fixed	
7	ANTLR	Parsing LUA long comment as short comment	Fixed	
8	ANTLR	Ambiguity in URL grammar	Fixed	
9	ANTLR	Underscore not allowed in XML NameStartChar	Fixed	
10	CJSON	Fail to reject invalid escape character	Confirmed	4
11	FAST-XML-PARSER	Fail to reject multiple root nodes	Confirmed	
12	FAST-XML-PARSER	Validation of invalid XML declarations	Confirmed	
13	FAST-XML-PARSER	Parsing an invalid XML element	Confirmed	
14	PARSON	Accepting invalid array	Rejected	2
15	PARSON	EOF not enforced	Rejected	
16	CJSON	Accepting invalid integers	Reported	2
17	PY-LUA-PARSER	Literal string gets parsed as LUA code	Reported	

Discrepancy Bugs

Discrepancies found for both SUTs:

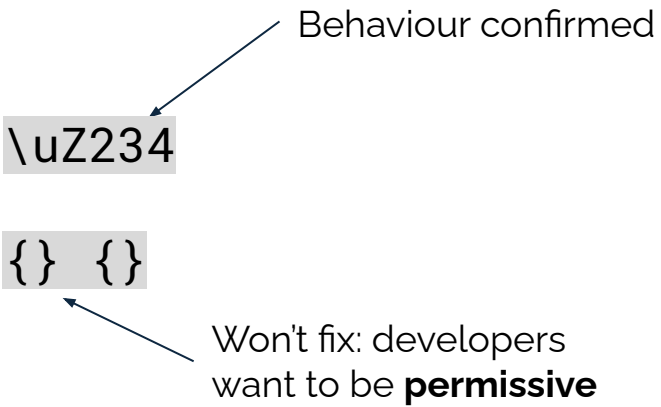
- cJSON accepts invalid unicode: `\uZ234`
- Parson accepts invalid JSON: `{} {}`

Discrepancy Bugs

Discrepancies found for both SUTs:

- cJSON accepts invalid unicode: `\uZ234`

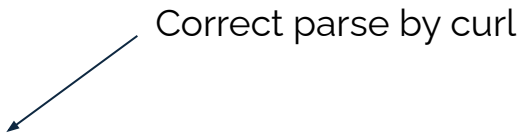
- Parson accepts invalid JSON: `{} {}`



Discrepancy Bug: CVE-2024-38428

Discrepancy found in URL SUTs:

curl "**`http://a;bc@xyz`**"
curl: (6) Could not resolve host: xyz



Discrepancy Bug: CVE-2024-38428

Discrepancy found in URL SUTs:

curl "**`http://a;bc@xyz`**"
curl: (6) Could not resolve host: xyz

Correct parse by curl

wget "**`http://a;bc@xyz`**"
wget: unable to resolve host address ‘a;bc@xyz’

Userinfo incorrectly parsed as hostname

What did we learn

1. Grammar mutation finds discrepancies
2. Edge test cases can reveal bugs

What did we learn

1. Grammar mutation finds discrepancies
2. Edge test cases can reveal bugs

But ..

What did we learn

1. Grammar mutation finds discrepancies
2. Edge test cases can reveal bugs

But ..

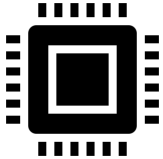
Exploration is still driven by grammar!

Repair-Driven Greybox Fuzzing

Bachir Bendrissou, Cristian Cadar, Alastair Donaldson

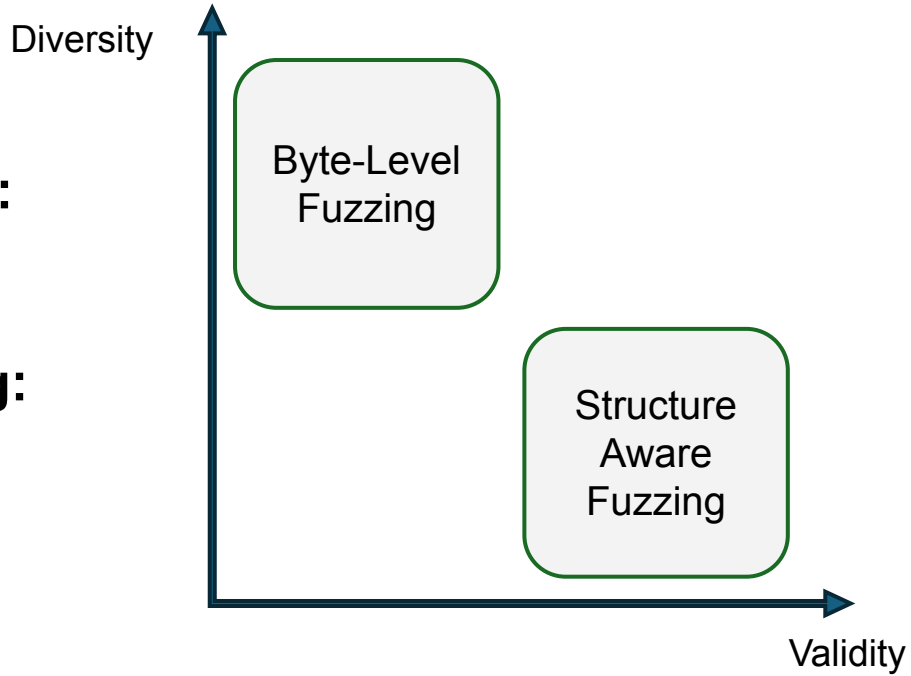
Imperial College London

under review at ISSTA 2026

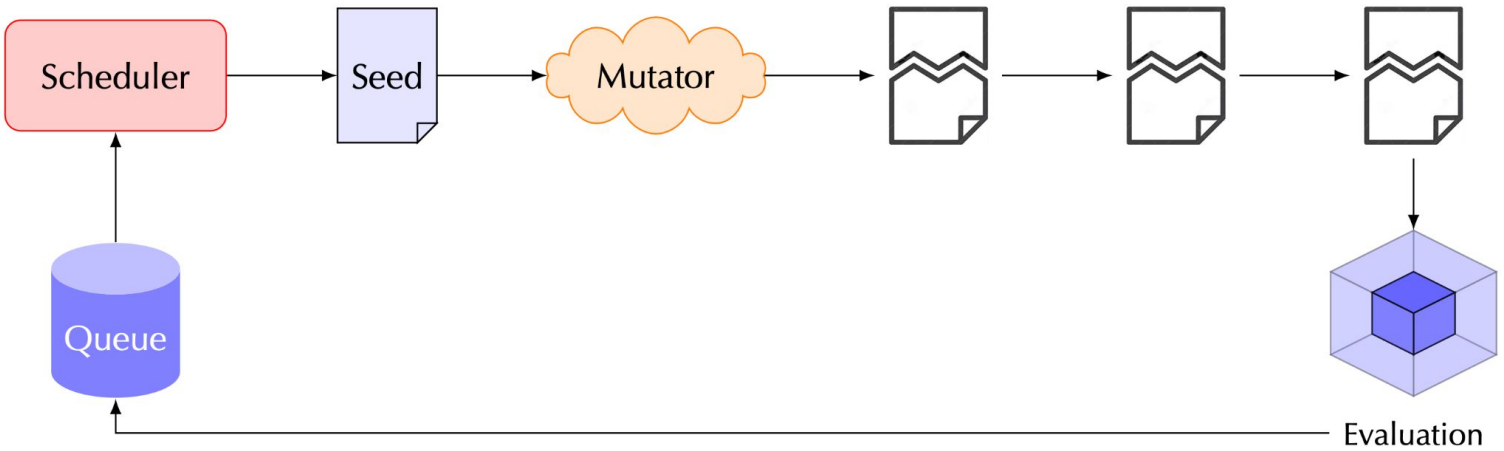


State of the Art

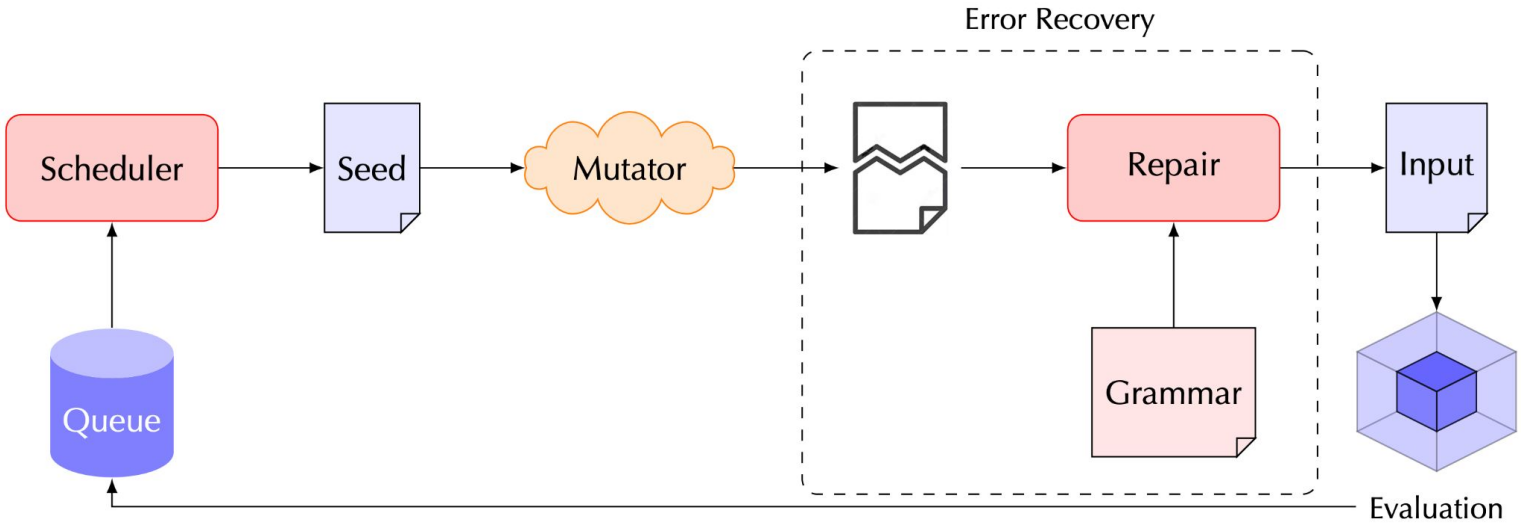
1. **Mutation-based fuzzing:**
   diverse tests, but invalid
2. **Grammar-based fuzzing:**
   valid tests, but biased



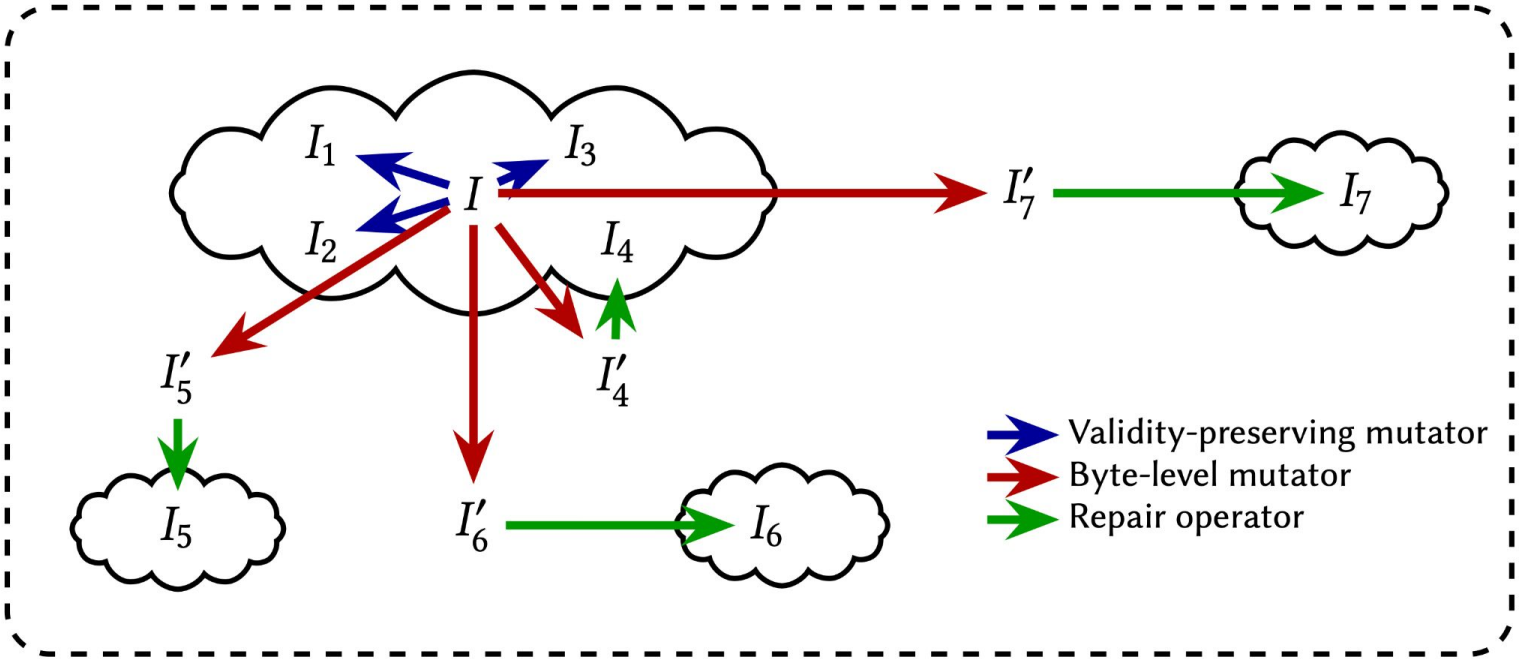
State of the Art: American Fuzzy Lop (AFL)



Proposed Solution: RepairFuzz



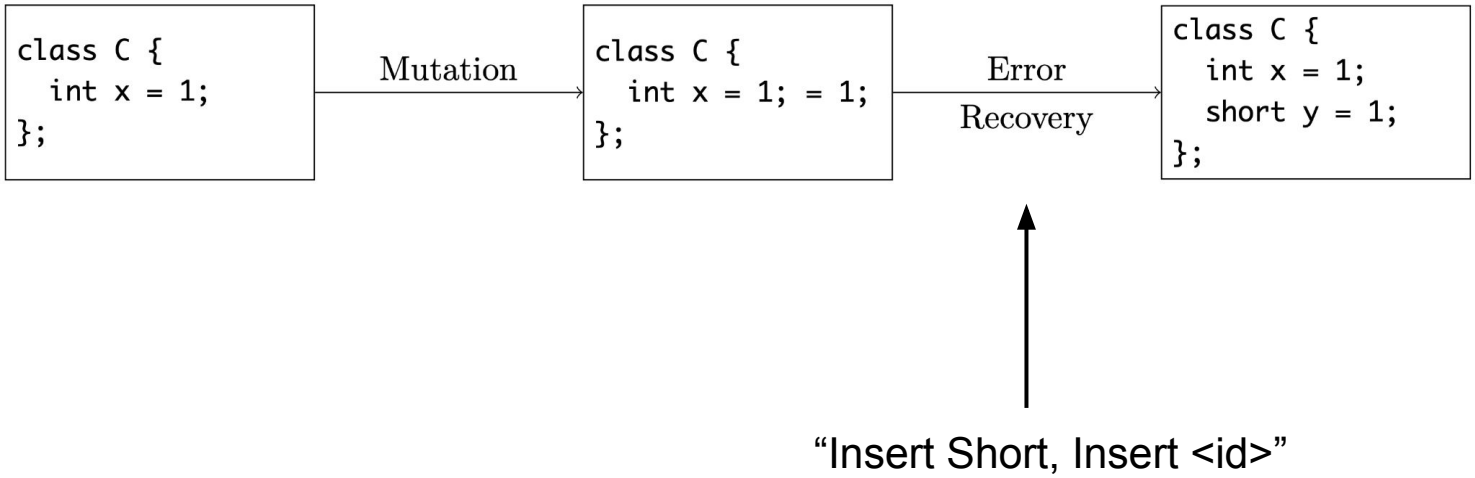
Proposed Solution: RepairFuzz



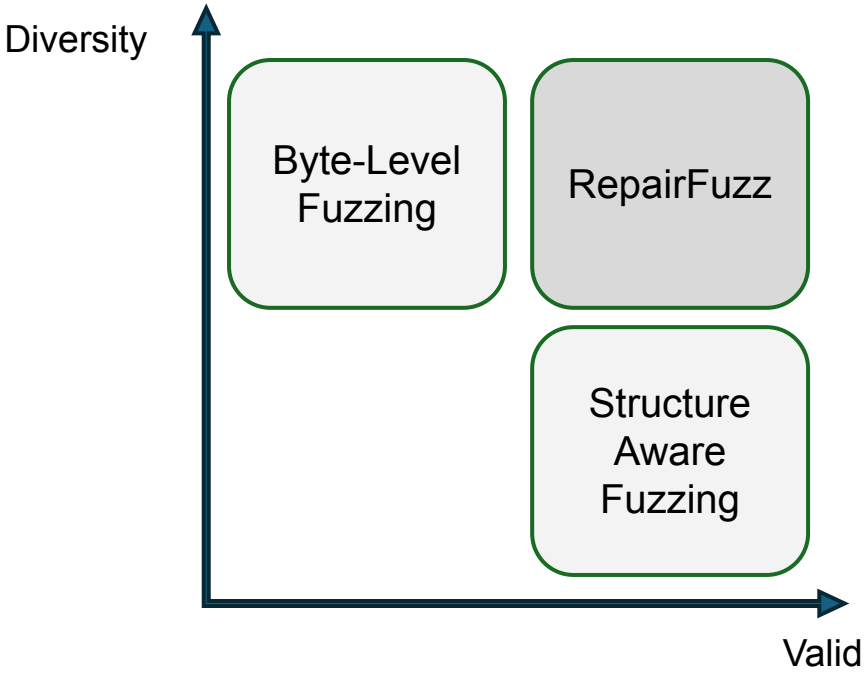
Automatic Error Recovery

1. Originally used to report programming errors and suggest fixes
2. Search-based approach
3. Highly precise and efficient

Automatic Error Recovery



★ *The Differentiating Factor*



Produce highly diverse test cases while maintaining structure validity

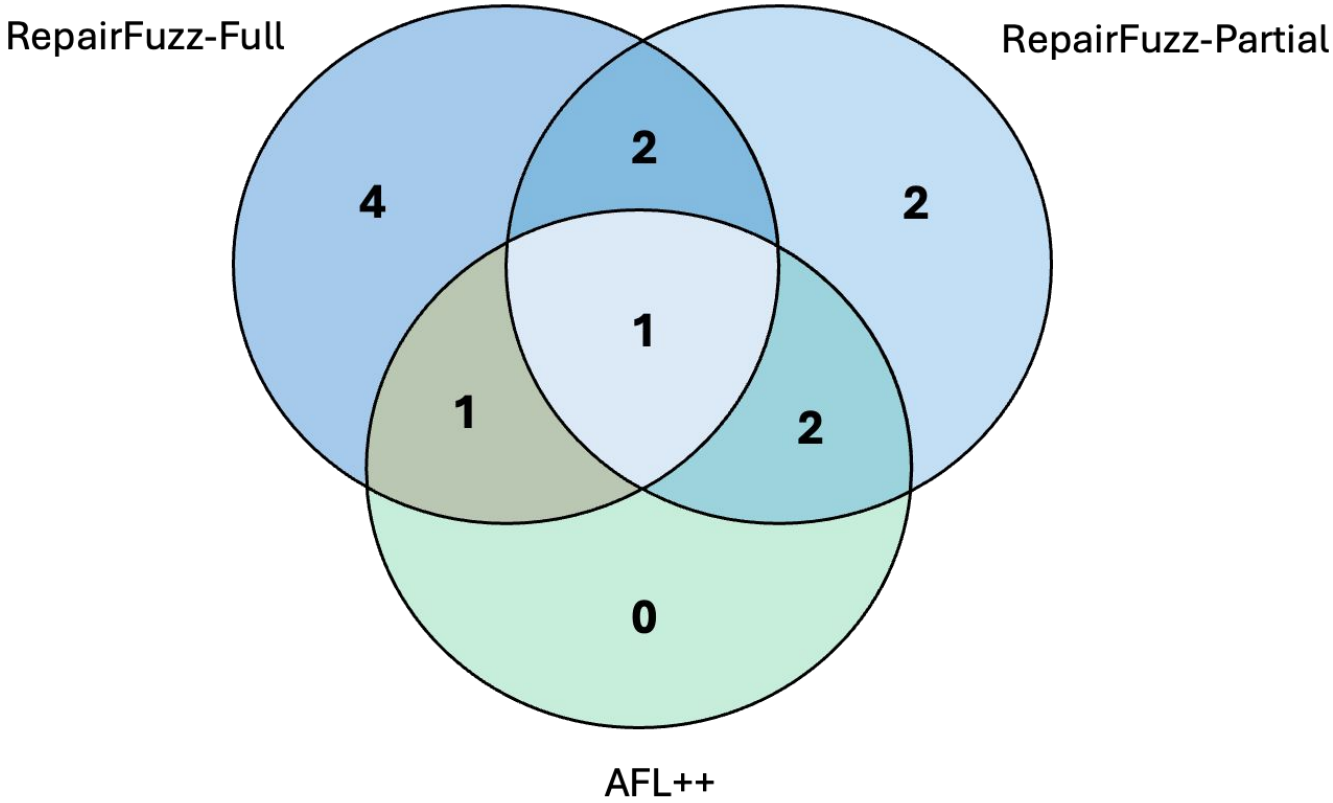
Evaluation: Systems Under Test

Program	Input format	Version	LOC
Lua	Lua	5.5.0	36K
LuaJIT	Lua	2.1	101K
Ruby	Ruby	3.3.5	2,054K
mRuby	Ruby	3.3.0	114K
PHP	PHP	8.5.0	1,828K
Chakracore	JavaScript	1.13.0	824K

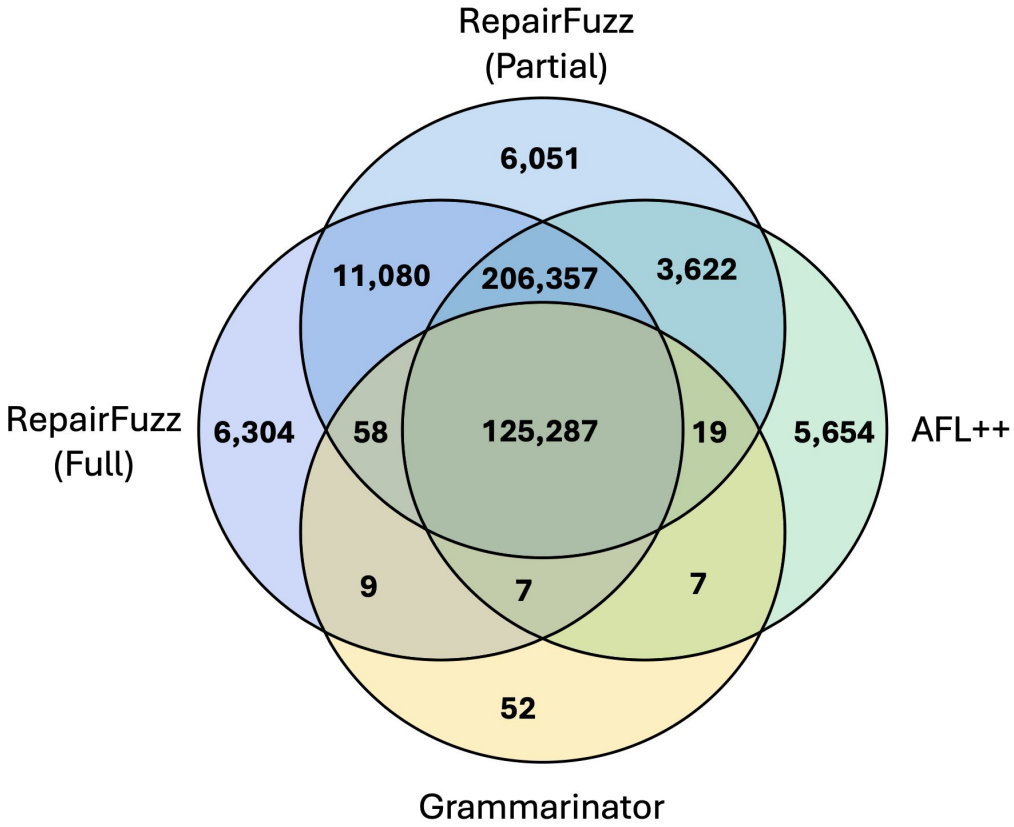
Evaluation Plan: Tools

1. **Grammarinator:** Grammar-based test generator
2. **Nautilus:** Greybox grammar-based test generator
3. **AFL++:** Imports Nautilus inputs as seed corpus, and performs byte-level mutations
4. **RepairFuzz-Full:** Imports Nautilus inputs as seed corpus, performs byte-level mutations, and repairs every mutated input
5. **RepairFuzz-Partial:** Only repairs mutated inputs 50% of the time

Results: Bugs found



Results: Unique Code Coverage



Original (valid)

```
<?php
$a = pack("AAAAAAAAAAAA", @1,2@,3,4,5,6,7,8,9,10,11,12);
unpack('h2147483647', $a);
?>
```

Byte-level mutation

Mutated (invalid)

```
<?php
$a = pack("AAAAAAAAAAAA", 1,2,3,4,5,6,7,8,9,10,11,12);
unpack(pack('h2147483647', $a);
?>
```

Grammar-based repair

Repaired (valid)

```
<?php
$a = pack("AAAAAAAAAAAA", 1,2,3,4,5,6,7,8,9,10,11,12);
unpack(pack('h2147483647', $a));
?>
```

Conclusion

- Grammars and implementations disagree.
- Near-valid inputs are highly valuable.
- Grammar Mutation produces **edge-case** inputs
- Validity alone isn't enough — **diversity** matters
- RepairFuzz achieves both: valid + diverse inputs

Thank you

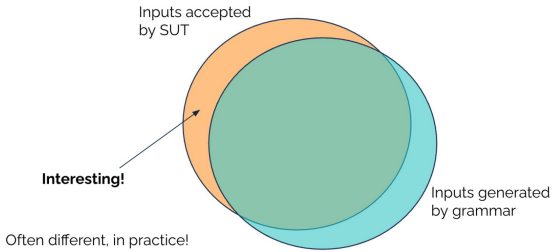
Example: mutating JSON

```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' HEX HEX HEX HEX ;
```

```
json
   : value (obj | EOF) ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value*)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE) ;
UNICODE
   : 'u' (STRING | HEX) HEX HEX HEX ;
```

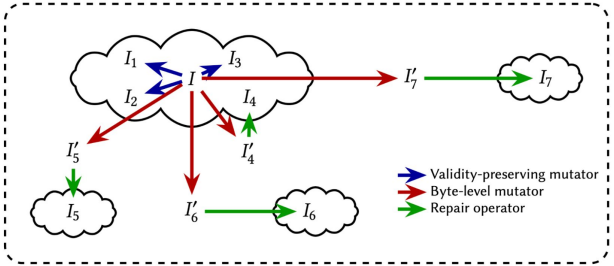
26

Grammars vs. SUTs



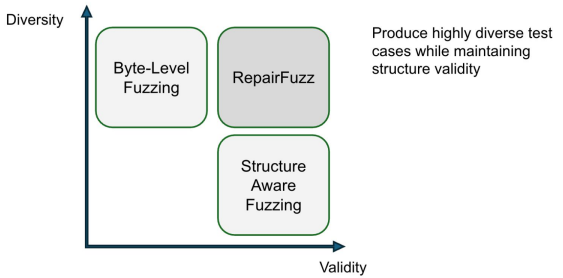
13

Proposed Solution: RepairFuzz



43

★ ***The Differentiating Factor***



46