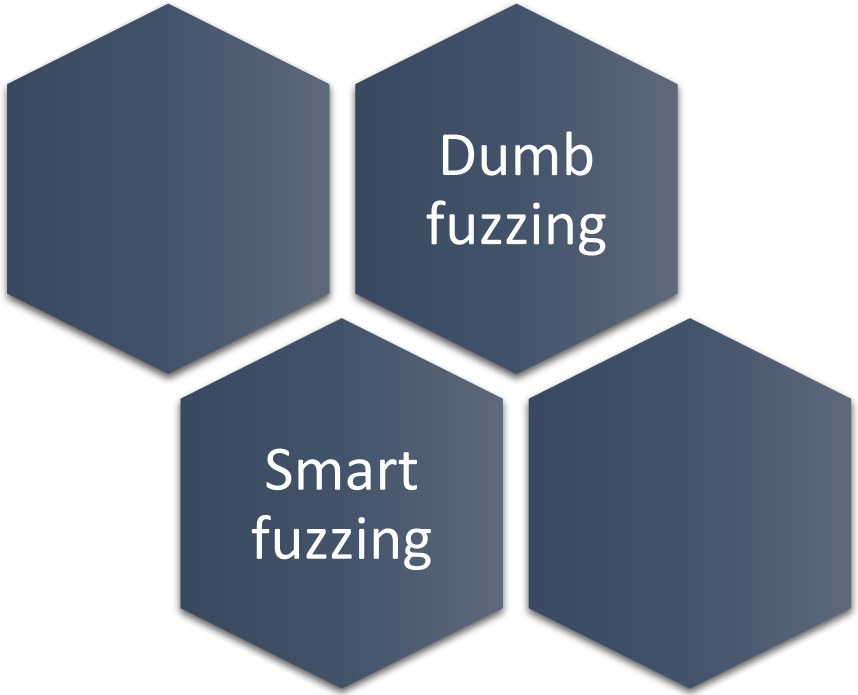
Grammar Mutation for Testing Input Parsers

Bachir Bendrissou, Cristian Cadar, Alastair F. Donaldson

Imperial College London

Fuzzing approaches



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**?

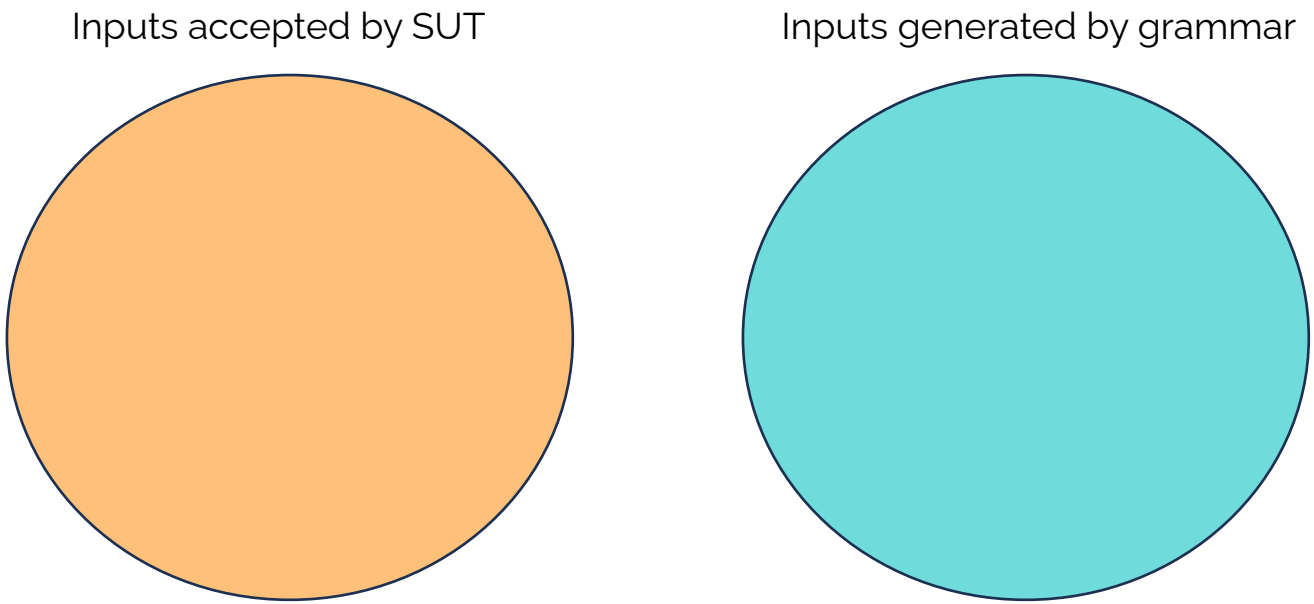
Generates **valid** inputs

Great! Valid inputs go **deep**

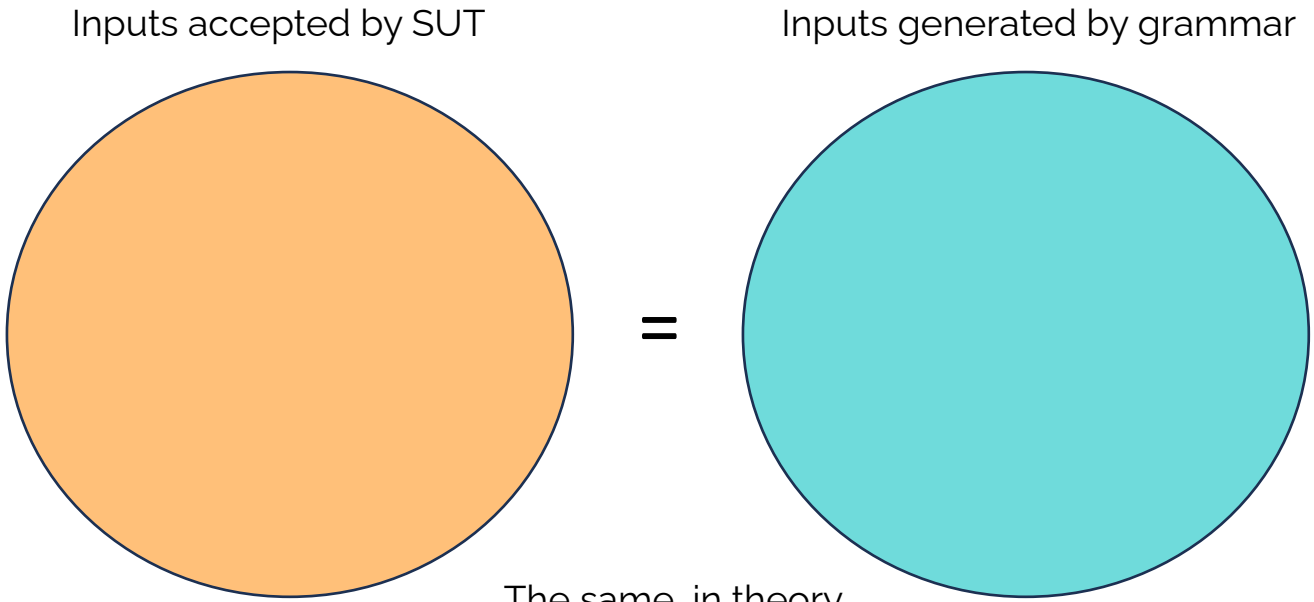
But…

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

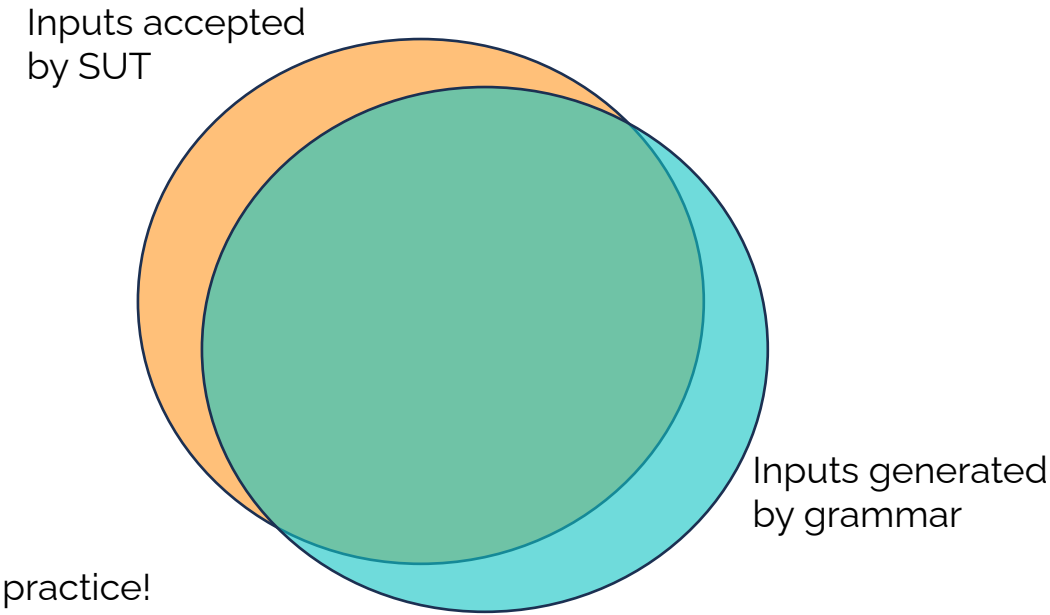
Grammars vs. SUTs



Grammars vs. SUTs

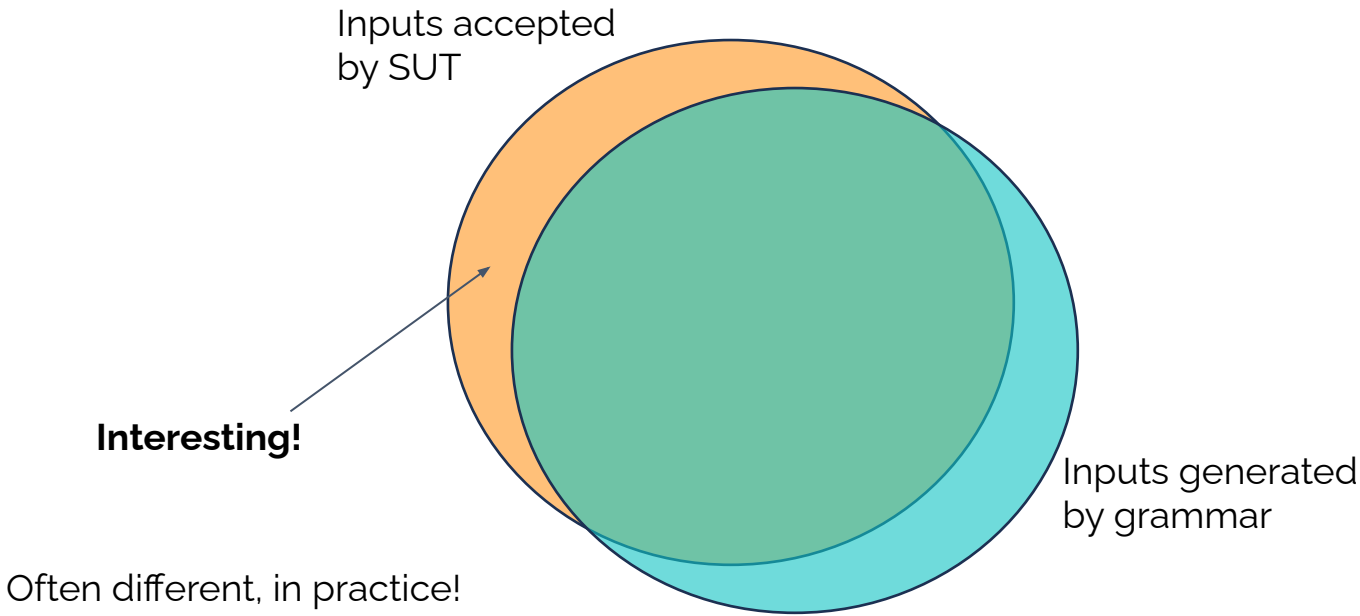


Grammars vs. SUTs



Often different, in practice!

Grammars vs. SUTs

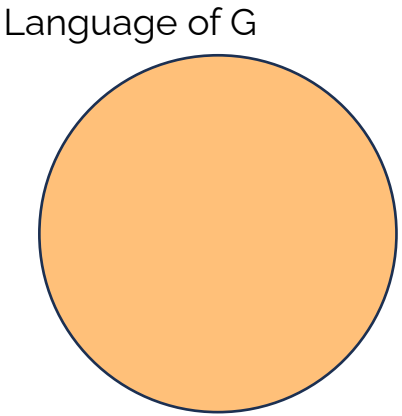


Often different, in practice!

Our idea: **grammar mutation**

Given:

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

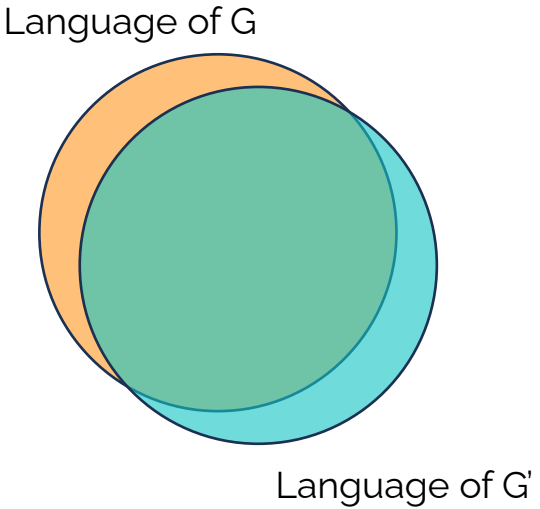


Our idea: **grammar mutation**

Given:

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

Mutate G to get **mutant grammar** G’



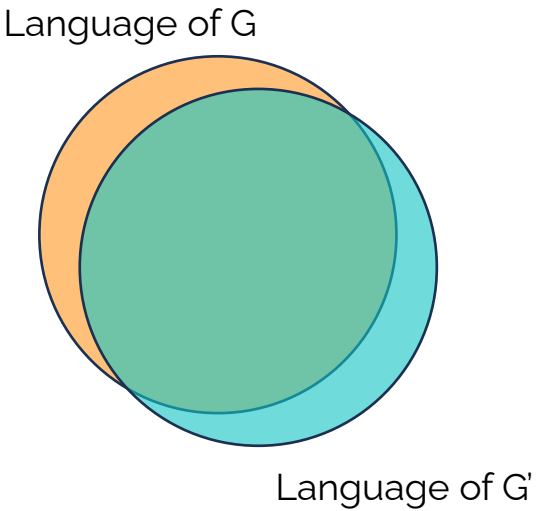
Our idea: **grammar mutation**

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**

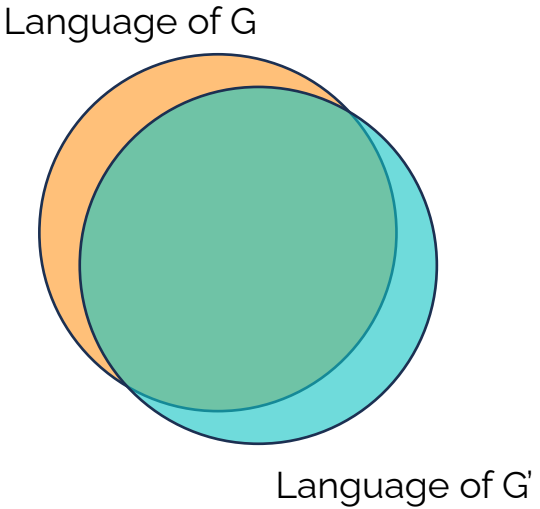
Given:

- 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**

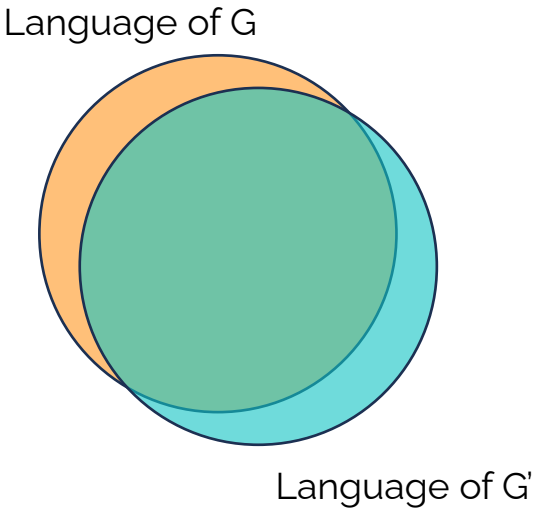
Given:

- 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

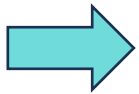
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE)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

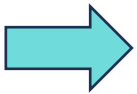
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE)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

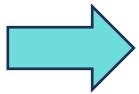
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE)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

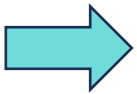
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE)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

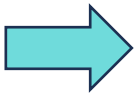
```
json
   : value EOF ;
obj
   : '{' pair (',' pair)* '}'
   | '{' '}' ;
pair
   : STRING ':' value ;
arr
   : '[' value (',' value)* ']'
   | '[' ']' ;
value
   : STRING | NUMBER | obj | arr
   | 'true' | 'false' | 'null' ;
STRING
   : '"' (ESC | CHAR)* '"' ;
ESC
   : '\\' (["\\/bfnrt] | UNICODE)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

Preliminary results

Gmutator tool:

- mutates Antlr grammars
- Uses Grammarinator for fuzzing

Input format: JSON

SUTs: cJSON, Parson

Preliminary results

Discrepancies found for both SUTs:

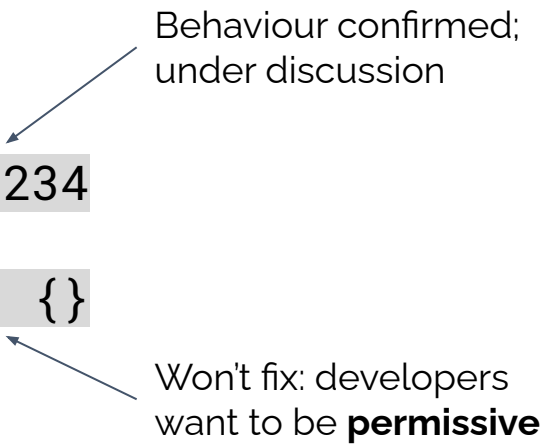
- cJSON accepts invalid unicode: `ur234`
- Parson accepts invalid JSON: `{} {}`

Preliminary results

Discrepancies found for both SUTs:

- cJSON accepts invalid unicode: `ur234`

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



Preliminary results

Discrepancies found for both SUTs:

- cJSON accepts invalid unicode: `ur234`

Behaviour confirmed; under discussion

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

Won't fix: developers want to be **permissive**

Modest improvements in **branch coverage** over Grammarinator baseline

- cJSON: 24% → 26%
- Parson: 19% → 22%

Results for inputs generated in 1 hour

Planned evaluation

Input formats

JSON

Lua

URL

XML

Planned evaluation

Input formats

JSON

Lua

URL

XML

SUTs

cJSON
Parson
simdjson

luac
LuaJIT
py-lua-parser

aria2
curl
wget

fast-xml-parser
libxml2
pugixml

Planned evaluation

Input formats

JSON

Lua

URL

XML

SUTs

cJSON
Parson
simdjson

luac
LuaJIT
py-lua-parser

aria2
curl
wget

fast-xml-parser
libxml2
pugixml

Baselines:

- Grammarinator
- Grammarinator + AFL-style mutation

24 hour runs

3 repeat runs per configuration

Research questions

RQ1: Does Gmutator find discrepancies?

How frequently?

Research questions

RQ1: Does Gmutator find discrepancies?

How frequently?

RQ2: What are the reasons for discrepancies? Are they intentional?

Will developers care?

Research questions

RQ1: Does Gmutator find discrepancies?

How frequently?

RQ2: What are the reasons for discrepancies? Are they intentional?

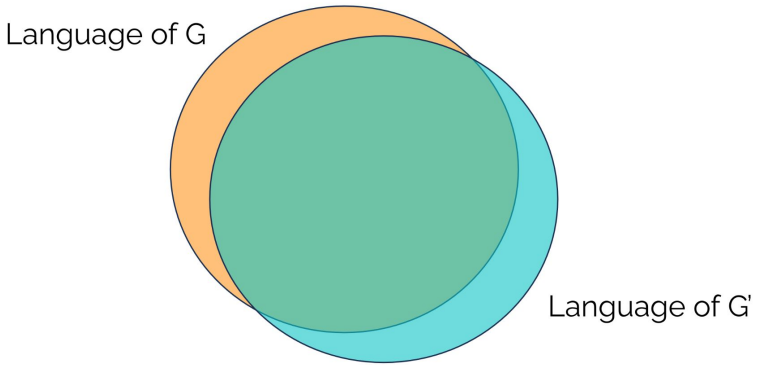
Will developers care?

RQ3: Does fuzzing with Gmutator yield improvements in code coverage?

Does it lead to discovery of new crashes?

Thank you

Grammar mutation



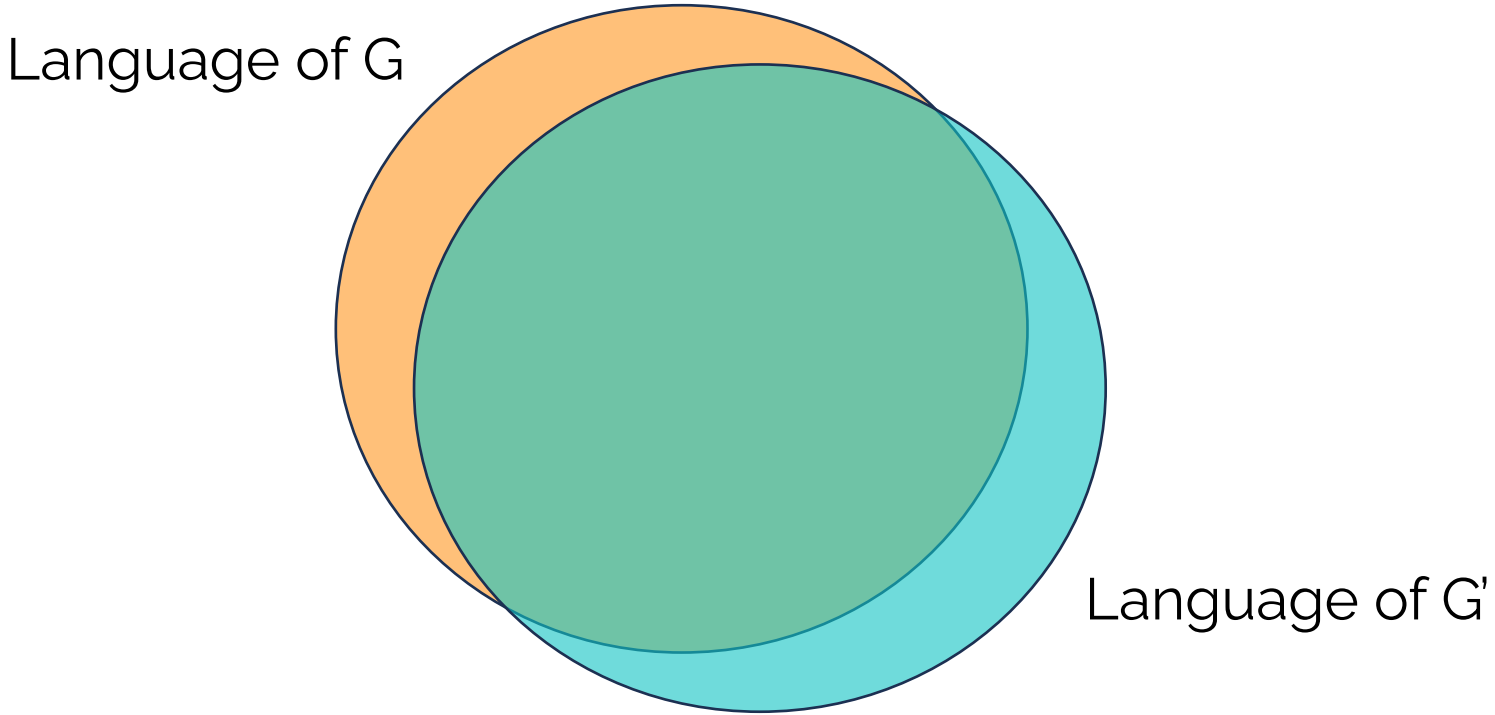
Gmutator

```
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 ;
```

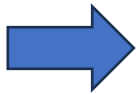
https://srg.doc.ic.ac.uk/projects/gmutator

Grammar mutation



Gmutator

```
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 ;
```