Mining Grammars

Andreas Zeller + Matthias Höschele
Center for IT-Security, Privacy and Accountability
Saarland University

@AndreasZeller
CISPA Saarbrücken
JavaScript Grammar

If Statement

IfStatementfull ⇒
  if ParenthesizedExpression Statementfull
| if ParenthesizedExpression Statementfull else Statementfull
IfStatemennosShortIf ⇒ if ParenthesizedExpression Statementfull else Statementfull

Switch Statement

SwitchStatement ⇒
  switch ParenthesizedExpression { }
| switch ParenthesizedExpression { CaseGroups LastCaseGroup }
A Generated Input

```javascript
1 var haystack = "foo";
2 var re_text = "^foo";
3 haystack += "x";
4 re_text += "(x)";
5 var re = new RegExp(re_text);
6 re.test(haystack);
7 RegExp.input = Number();
8 print(RegExp.$1);
```
A Generated Input

```javascript
1 var haystack = "foo";
2 var re_text = "^foo";
3 haystack += "x";
4 re_text += "(x)";
5 var re = new RegExp(re_text);
6 re.test(haystack);
7 RegExp.input = Number();
8 print(RegExp.$1);
```
Fuzzing JavaScript

# defects

Mozilla TI

Google V8
(Chrome 1.0 Beta)

Mozilla TM
(Firefox 4 Beta)

# days
Fuzzing JavaScript

- Mozilla TI
- Google V8 (Chrome 1.0 Beta)
- Mozilla TM (Firefox 4 Beta)

18 Chromium Security Rewards
12 Mozilla Security Bug Bounty Awards in 9 months
Fuzzing JavaScript

© days

# defects

Mozilla TI

Google V8
(Chrome 1.0 Beta)

Mozilla TM
(Firefox 4 Beta)

US$ 50,000+ in first four weeks
Creating Grammars

If Statement

\[
\text{IfStatement}^{\text{full}} \Rightarrow \\
\text{if ParenthesizedExpression Statement}^{\text{full}} \\
\mid \text{if ParenthesizedExpression Statement}^{\text{noShortIf}} \text{ else Statement}^{\text{full}} \\
\text{IfStatement}^{\text{noShortIf}} \Rightarrow \text{if ParenthesizedExpression Statement}^{\text{noShortIf}} \text{ else Statement}^{\text{noShortIf}}
\]

Switch Statement

\[
\text{SwitchStatement} \Rightarrow \\
\text{switch ParenthesizedExpression \{ \} } \\
\mid \text{switch ParenthesizedExpression \{ CaseGroups LastCaseGroup \} }
\]

CaseGroups ⇒

«empty»

| CaseGroups CaseGroup

CaseGroup ⇒ CaseGuards BlockStatementsPrefix

LastCaseGroup ⇒ CaseGuards BlockStatements

CaseGuards ⇒

CaseGuard

| CaseGuards CaseGuard

CaseGuard ⇒
Creating Grammars

If Statement

\[ \text{IfStatement}^{\text{full}} \Rightarrow \]
\[ \text{if } \text{ParenthesizedExpression} \]
\[ \text{if } \text{ParenthesizedExpression} \]
\[ \text{IfStatement}^{\text{noShortIf}} \Rightarrow \]

Switch Statement

\[ \text{SwitchStatement} \Rightarrow \]
\[ \text{switch } \text{ParenthesizedExpression} \]
\[ \text{switch } \text{ParenthesizedExpression} \]
\[ \text{CaseGroups} \Rightarrow \]
\[ \text{«empty»} \]
\[ \text{CaseGroups CaseGroup} \Rightarrow \]
\[ \text{Case} \]
\[ \text{LastCaseGroup} \Rightarrow \]
\[ \text{CaseGuard} \]
\[ \text{CaseGuards} \Rightarrow \]
\[ \text{CaseGuard} \]
\[ \text{CaseGuards CaseGuard} \]
\[ \text{CaseGuard} \Rightarrow \]
Learning Grammars

Höschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

Hö schele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

Höschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Höschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

Höschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

Höschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Hö schele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

http://user:pass@www.google.com:80/path

Program
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol

www.google.com – host name
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
www.google.com – host name
80 – port
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
www.google.com – host name
80 – port
user pass – login
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
www.google.com – host name
80 – port
user pass – login
path – page request
Learning Grammars

http://user:pass@www.google.com:80/path

- http – protocol
- www.google.com – host name
- 80 – port
- user pass – login
- path – page request
- ://: @ : / – terminals
Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
- page request
- terminals

processed in different functions

stored in different variables
Tracking Input

We track input characters throughout program execution:

1. Dynamic tainting labels all characters read (and derived values) with their origin

2. Recognizing inputs checks string variables whether they hold input fragments (simpler)
Tracking in Python

# We store individual variable/value pairs here
global the_values
the_values = {}  

# The current input string
global the_input
the_input = None

# We record all string variables and values occurring during execution
def traceit(frame, event, arg):
    global the_values
    variables = frame.f_locals.keys()
    for var in variables:
        value = frame.f_locals[var]
        # Save all non-trivial string values that also occur in the input
        if type(value) == type('') and len(value) >= 2 and value in the_input:
            the_values[var] = value

    return traceit

the_input = "...
sys.settrace(traceit)
program_under_test(the_input)
input = 'http://user:pass@www.google.com:80/path#ref'
urlparse(input) ->
    fragment = 'ref'
    url = '/path'
    path = '/path'
    scheme = 'http'
    netloc = 'user:pass@www.google.com:80'
Grammar Inference

- Start with grammar $\text{START} ::= \text{input}$

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $\text{VAR}$
  2. Add a new rule $\text{VAR} ::= \text{value}$
Grammar Inference

- Start with grammar $START ::= \text{input}$

$START ::= \text{http://user:pass@www.google.com:80/path#ref}$
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR
  2. Add a new rule $VAR ::= value

$START ::= http://user:pass@www.google.com:80/path#ref

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
netloc = 'user:pass@www.google.com:80'
Grammar Inference

• For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR
  2. Add a new rule $VAR ::= value

$START ::= http://user:pass@www.google.com:80/path#ref

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
netloc = 'user:pass@www.google.com:80'
Grammar Inference

• For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= \text{http://user:pass@www.google.com:80/path}\#ref$

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
netloc = 'user:pass@www.google.com:80'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= http://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= http://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= http://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80

fragment = 'ref'
url = '/path'
path = '/path'
scheme = 'http'
Grammar Inference

• For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR
  2. Add a new rule $VAR ::= value

$START ::= $SCHEME://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http

fragment = 'ref'
url = '/path'
path = '/path'
Grammar Inference

• For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http

fragment = 'ref'
url = '/path'
path = 'path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC/path#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http

fragment = 'ref'
url = '/path'
path = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path

fragment = 'ref'
url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$$START ::= \text{SCHEME://NETLOC}PATH\#ref$$
$$NETLOC ::= user:pass@www.google.com:80$$
$$SCHEME ::= http$$
$$PATH ::= /path$$

fragment = 'ref'
url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path

fragment = 'ref'
url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#$FRAGMENT
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path
$FRAGMENT ::= ref

url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#$FRAGMENT
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path
$FRAGMENT ::= ref

url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#$FRAGMENT
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path
$FRAGMENT ::= ref

url = '/path'
Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR$
  2. Add a new rule $VAR ::= \text{value}$

$START ::= \text{$SCHEME://$NETLOC$PATH#$FRAGMENT}$
$NETLOC ::= \text{user:pass@www.google.com:80}$
$SCHEME ::= \text{http}$
$PATH ::= \text{$URL}$
$FRAGMENT ::= \text{ref}$
$URL ::= /path$
Grammar Expansions

# Obtain a grammar for a specific input
def get_grammar(input):
    # Here's our initial grammar
    grammar = {
        "$START": [input]
    }

    # We obtain a mapping of variables to values
    global the_input
    the_input = input

    global the_values
    the_values = {}

    sys.settrace(traceit)
o = urlparse(the_input)
sys.settrace(None)

    # Now for each (VAR, VALUE) found:
    # 1. We search for occurrences of VALUE in the grammar
    # 2. We replace them by $VAR
    # 3. We add a new rule $VAR -> VALUE to the grammar
    while True:
        new_rules = []
        for var in the_values.keys():
            value = the_values[var]
            # Replace all occurrences of value in grammar
            for key in grammar.keys():
                if key in value:
                    grammar[key] = grammar[key].replace(value, var)
            # Add new rule
            grammar[var] = value
        if not new_rules:
            break
# Now for each (VAR, VALUE) found:
# 1. We search for occurrences of VALUE in the grammar
# 2. We replace them by $VAR
# 3. We add a new rule $VAR -> VALUE to the grammar

while True:
    new_rules = []
    for var in the_values.keys():
        value = the_values[var]
        for key in grammar.keys():
            repl_alternatives = grammar[key]
            for j in range(0, len(repl_alternatives)):
                repl = repl_alternatives[j]
                if value in repl:
                    # Found variable value in some grammar nonterminal
                    # Replace value by nonterminal name
                    alt_key = nonterminal(var)
                    repl_alternatives[j] = repl.replace(value, alt_key)
                    new_rules = new_rules + [(var, alt_key, value)]

    if len(new_rules) == 0:
        break  # Nothing to expand anymore

    for (var, alt_key, value) in new_rules:
        # Add new rule to grammar
        grammar[alt_key] = [value]

        # Do not expand this again
        del the_values[var]

return grammar
Merging Grammars

- Multiple inputs yield multiple grammars
- Merge these grammars to obtain alternatives
Demo
# Merge two grammars $G_1$ and $G_2$

```python
def merge_grammars(g1, g2):
    merged_grammar = g1
    for key2 in g2.keys():
        repl2 = g2[key2]
        key_found = False
        for key1 in g1.keys():
            repl1 = g1[key1]
            for repl in repl2:
                if key1 == key2:
                    key_found = True
                    if repl not in repl1:
                        # Extend existing rule
                        merged_grammar[key1] = repl1 + [repl]
                if not key_found:
                    # Add new rule
                    merged_grammar[key2] = repl2
    return merged_grammar
```

Merging Grammars
Merged Grammars

'http://www.st.cs.uni-saarland.de/zeller#ref' ->
$START ::= $SCHEME://$NETLOC$URL#$FRAGMENT
$SCHEME ::= http
$NETLOC ::= www.st.cs.uni-saarland.de
$URL ::= $PATH
$PATH ::= /zeller
$FRAGMENT ::= ref

'https://www.cispa.saarland:80/bar' ->
$START ::= $SCHEME://$NETLOC$URL
$SCHEME ::= https
$NETLOC ::= www.cispa.saarland:80
$URL ::= $PATH
$PATH ::= /bar
Merged Grammars

Merged grammar ->
$url ::= $path
$start ::= $scheme://$netloc$url#$fragment | $scheme://$netloc$url | $scheme://$netloc$url?
$query#$fragment
$path ::= /zeller | /bar
$query ::= q=r
$netloc ::= www.st.cs.uni-saarland.de | www.cispa.saarland:80 | foo@google.com:8080
$fragment ::= ref | ref2
$scheme ::= http | https
Fuzzing

Fuzzing ->
https://www.cispa.saarland:80/zeller
https://www.cispa.saarland:80/bar#ref
http://www.st.cs.uni-saarland.de/zeller#ref2
http://www.cispa.saarland:80/bar#ref
https://www.st.cs.uni-saarland.de/zeller#ref
http://foo@google.com:8080/bar
http://www.cispa.saarland:80/bar#ref
https://www.st.cs.uni-saarland.de/bar#ref2
http://www.st.cs.uni-saarland.de/zeller#ref
...
Fuzzing File Formats

List of file formats

From Wikipedia, the free encyclopedia

This is a dynamic list and may never be able to satisfy particular standards for completeness. You can help by expanding it with reliably sourced entries.

See also: List of filename extensions

This is a list of file formats used by computers, organized by type. Filename extensions are usually noted in parentheses if they differ from the format name or abbreviation. Many operating systems do not limit filenames to a single extension shorter than 4 characters, as was common with some operating systems that supported the FAT file system. Examples of operating systems that do not impose this limit include Unix-like systems. Also, Microsoft Windows NT, 95, 98, and Me do not have a three character limit on extensions for 32-bit or 64-bit applications on file systems other than pre-Windows 95/Windows NT 3.5 versions of the FAT file system. Some filenames are given extensions longer than three characters.

Some file formats may be listed twice or more. An example is the .b file.
List of file formats

This is a dynamic list and may never be able to satisfy particular standards for completeness. You can help by expanding it with reliably sourced entries.

See also: List of filename extensions

This is a list of file formats used by computers, organized by type. Filename extensions are usually noted in parentheses if they differ from the format name or abbreviation. Many operating systems do not limit filenames to a single extension shorter than 4 characters, as was common with some operating systems that supported the FAT file system. Examples of operating systems that do not impose this limit include Unix-like systems. Also, Microsoft Windows NT 95, 98, and Me do not have a three character limit on extensions for 32-bit or 64-bit applications on file systems other than pre-Windows 95/Windows NT 3.5 versions of the FAT file system. Some filenames are given extensions longer than three characters.

Some file formats may be listed twice or more. An example is the .b file.

1. Archive and compressed
   1.1 Physical recordable media archiving
2. Computer-aided Design
   2.1 Computer-aided design (CAD)
3. Electronic design automation (EDA)
Fuzzing File Formats
AUTOGRAM for Java

- Grammar Inference for Java programs
- First technique to mine input grammars from programs
- Uses dynamic tainting for deep structure recognition
- Uses active learning to identify repetitions and optional parts
- Has predefined lexical items for frequently occurring input fragments
URLs

http://user:pass@www.google.com:80/command
?foo=bar&lorem=ipsum#fragment

SPEC ::= STRING [[‘?’ [QUERY]] [‘#’ [REF]]]
STRING ::= PROTOCOL ‘://’ AUTHORITY [PATH]
AUTHORITY ::= [USERINFO ‘@’] HOST
    [‘:’ PORT]
PROTOCOL ::= ‘http’
USERINFO ::= ‘user:pass’
HOST ::= HOSTNAME
PORT ::= DIGITS
PATH ::= ABSOLUTEPATH
QUERY ::= ALPHANUMWITHSPECIALS
REF ::= ALPHANUMWITHSPECIALS
INI Files

[Application]
Version = 1
WorkingDir = mydir

[User]
User = Bob
Password = 12345

LOAD ::= LINE_FIRST LINE* LINE1
LINE_FIRST ::= LINE2_FIRST \n'
LINE2_FIRST ::= '[_ SECTION _]''
SECTION ::= ALPHANUMWHITEPRINT
LINE ::= LINE2 \n'
LINE2 ::= '(_ SECTION _)' |
    KEY ' _ * ' = ' (_ * [VALUE]]'
KEY ::= ALPHANUMWHITEPRINT
VALUE ::= ALPHANUMWHITEPRINT
LINE1 ::= KEY ' _ * ' = ' (_ _ [VALUE]]
JSON Input

```json
{
  "glossary": {
    "title": "example glossary",
    "GlossSeeAlso": [
      "GML", "XML"
    ],
    "bool1": true,
    "bool2": false,
    "number1": 2349872,
    "number2": -45242,
    "number3": 2349.872,
    "number4": -98.72,
    "empty": null,
    "number5": 2372e71,
    "number6": 123e-31,
    "number7": 23.72e71,
    "number8": 12.83e-33
  }
}
```
JSON Input

MAIN ::= VALUE
VALUE ::= STRING | FALSE | TRUE | OBJECT | ARRAY | NULL | NUMBER
STRING ::= '"' HASH '"'
HASH ::= ALPHANUMWITHSPECIALWHITESPACES
FALSE ::= 'false'
TRUE ::= 'true'
OBJECT ::= '{'
    [ STRINGINTERNAL ':' VALUE
    (',' STRINGINTERNAL ':' VALUE)* ] '}'
STRINGINTERNAL ::= '"' HASH '"'
ARRAY ::= '[' [ VALUE ( ',' VALUE )* ] ']
NULL ::= 'null'
NUMBER ::= INTEGER [[FRACTION] [EXPONENT]]
EXPONENT ::= 'e' ['-''] POSITIVEINTEGER
FRACTION ::= '.' POSITIVEINTEGER
AUTOGRAM Grammars
AUTOGRAM Grammars

• give insights into the structure of inputs
  → reverse engineering
  → writing tests
  → writing parsers
AUTOGRAM Grammars

• give insights into the structure of inputs
  → reverse engineering
  → writing tests
  → writing parsers

• first technique to mine input grammars from programs
AUTOGRAM Grammars

• give insights into the structure of inputs
  → reverse engineering
  → writing tests
  → writing parsers

• first technique to mine input grammars from programs

• fully automatic • scalable • practical
Mining Grammars

Andreas Zeller + Matthias Höschele
Center for IT-Security, Privacy and Accountability
Saarland University

@AndreasZeller
Grammar-Based Fuzzing

```javascript
var haystack = "foo";
var re_text = "~foo";
haystack += "x";
re_text += "(x)";
var re = new RegExp(re_text);
re.test(haystack);
RegExp.input = Number();
print(RegExp.$1);
```

---

Andreas Zeller + Matthias Höschele
Center for IT-Security, Privacy and Accountability
Saarland University

@AndreasZeller
Grammar-Based Fuzzing

<table>
<thead>
<tr>
<th>var</th>
<th>haystack = &quot;foo&quot;;</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>re_text = &quot;^foo&quot;;</td>
</tr>
<tr>
<td></td>
<td>haystack += &quot;x&quot;;</td>
</tr>
<tr>
<td></td>
<td>re_text += &quot;(x)&quot;;</td>
</tr>
<tr>
<td>var</td>
<td>re = new RegExp(re_text);</td>
</tr>
<tr>
<td></td>
<td>re.test(haystack);</td>
</tr>
<tr>
<td></td>
<td>RegExp.input = Number();</td>
</tr>
<tr>
<td></td>
<td>print(RegExp.$1);</td>
</tr>
</tbody>
</table>

Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
- page request
- terminals

Andreas Zeller + Matthias Höschele
Center for IT-Security, Privacy and Accountability
Saarland University

@AndreasZeller
Grammar-Based Fuzzing

```javascript
var haystack = "foo";
var re_text = "foo";
haystack += "x";
re_text += "(x)";
var re = new RegExp(re_text);
re.test(haystack);
RegExp.input = Number();
print(RegExp.$1);
```

Learning Grammars

```java
http://user:pass@www.google.com:80/path
```

Grammar Inference

- For each (\textit{var}, \textit{value}) we find during execution, where \textit{value} is a substring of \textit{input}:
  1. Replace all occurrences of \textit{value} by $VAR$
  2. Add a new rule $VAR ::= value$

$START ::= $SCHEME://$NETLOC$PATH#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path

fragment = 'ref'
url = '/path'

Matthias Höschele
Privacy and Accountability
University

asZeller
Grammar-Based Fuzzing

```java
var haystack = "foo";
var re_text = "~foo";
haystack += "x";
re_text += "(x)";
var re = new RegExp(re_text);
re. test (haystack);
RegExp. input = Number();
print(RegExp. $1);
```

Learning Grammars

```
http://user:pass@www.google.com:80/path
```

Grammar Inference

- For each (var, value) we find during execution, where value is a substring of input:
  1. Replace all occurrences of value by $VAR
  2. Add a new rule $VAR ::= value

```
$START ::= $SCHEME://$NETLOC$PATH#ref
$NETLOC ::= user:pass@www.google.com:80
$SCHEME ::= http
$PATH ::= /path

fragment = 'ref'
url = '/path'
```

AUTOGRAM for Java

- Grammar Inference for Java programs
- First technique to mine input grammars from programs
- Uses dynamic tainting for deep structure recognition
- Uses active learning to identify repetitions and optional parts
- Has predefined lexical items for frequently occurring input fragments
Grammar-Based Fuzzing

```
1 var haystack = "foo";
2 var re_text = "\"foo\"";
3 haystack += "x";
4 re_text += "(x)";
5 var re = new RegExp(re_text);
6 re.test(haystack);
7 RegExp.input = Number();
8 print(RegExp.$1);
```

Learning Grammars

```
http://user:pass@www.google.com:80/path
```

http://www.st.cs.uni-saarland.de/models/autogram

Grammar Inference

- For each \( (\text{var}, \text{value}) \) we find during execution, where \text{value} is a substring of \text{input}:
  1. Replace all occurrences of \text{value} by \$\text{VAR}
  2. Add a new rule \$\text{VAR} ::= \text{value}

**AUTOGRAM for Java**

- Grammar Inference for Java programs
- First technique to mine input grammars from programs
- Uses \textit{dynamic tainting} for deep structure recognition
- Uses \textit{active learning} to identify \textit{repetitions} and \textit{optional parts}
- Has \textit{predefined lexical items} for frequently occurring input fragments
Dynamic Grammar Inference in Java

Security Testing WS 2017/18
Matthias Höschele

https://www.st.cs.uni-saarland.de/models/autogram/
Learning Grammars

http://user:pass@www.google.com:80/path

Program
Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
- page request
- terminals

processed in different functions
stored in different variables
Dynamic Tainting

- Each value can be marked with a taint
- Taints are stored in shadow memory
- Programs are instrumented to propagate taint markers
- Trace taint values at specific events to observe data flow
Dynamic Tainting

Taints:
\[ a \quad b \quad \text{result} \]

```java
int a = read(file);
int b = read(file);
int result = a * 10;
result = result - b;
```
Dynamic Tainting

```c
int a = read(file);
int b = read(file);
int result = a * 10;
result = result - b;
```

Taints:
- $a$
- $b$
- result

$\rightarrow$
Dynamic Tainting

```c
int a = read(file);
int b = read(file);
int result = a * 10;
result = result - b;
```

Taints:
```
a  b  result
```
Dynamic Tainting

```c
int a = read(file);
int b = read(file);
int result = a * 10;
result = result - b;
```

Taints:
- `a`
- `b`
- `result`
Dynamic Tainting

```
int a = read(file)
int b = read(file)
int result = a * 10;
result = result - b;
```

Taints:

- `a`: Blue
- `b`: Green
- `result`: Blue
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

```java
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  |  
  param: protocol
  |  
  param: host
  |  
  param: port
  |  
  param: authority
  |  
  param: userinfo
  |  
  param: path
  |  
  param: query
  |  
  param: ref
```
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
    |__________________________________________|
param: protocol
    |__________________________________________|
    | http|
param: host
param: port
param: authority
param: userinfo
param: path
param: query
param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  |
param: protocol  | http
param: host      | www.google.com
param: port      |
param: authority |
param: userinfo  |
param: path      |
param: query     |
param: ref       |
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| param: protocol |
|-----------------|-----------------|-----------------|-----------------|
| http            |
| param: host     |
| www.google.com  |
| param: port     |
| port            |
| param: authority|
| authority       |
| param: userinfo |
| user:password   |
| param: path     |
| path            |
| param: query    |
| query           |
| param: ref      |
| ref             |
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | param: protocol
  | http
  | param: host
  | www.google.com
  | param: port
  | 80
  | param: authority
  | param: userinfo
  | user:password
  | param: path
  | /command
  | param: query
  | param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

- param: protocol
  - http
- param: host
  - www.google.com
- param: port
  - 80
- param: authority
- param: userinfo
  - user:password
- param: path
  - /command
- param: query
  - foo=bar&lorem=ipsum
- param: ref
  - fragment
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

<table>
<thead>
<tr>
<th>param: protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
</tr>
<tr>
<td>param: host</td>
</tr>
<tr>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>param: port</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>param: authority</td>
</tr>
<tr>
<td>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80</td>
</tr>
<tr>
<td>param: userinfo</td>
</tr>
<tr>
<td>user:password</td>
</tr>
<tr>
<td>param: path</td>
</tr>
<tr>
<td>/command</td>
</tr>
<tr>
<td>param: query</td>
</tr>
<tr>
<td>foo=bar&amp;lorem=ipsum</td>
</tr>
<tr>
<td>param: ref</td>
</tr>
<tr>
<td>fragment</td>
</tr>
</tbody>
</table>
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | http
param: protocol
  | http
param: host
  | www.google.com
param: port
  | 80
param: authority
  | user:password@www.google.com:80
param: userinfo
  | user:password
param: path
  | /command
param: query
  | foo=bar&lorem=ipsum
param: ref
  | fragment
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | http user:password@www.google.com:80/command foo=bar&lorem=ipsum fragment
param: protocol
  | http
param: host
  | www.google.com
param: port
  | 80
param: authority
  | user:password@www.google.com:80
param: userinfo
  | user:password
param: path
  | /command
param: query
  | foo=bar&lorem=ipsum
param: ref
  | fragment
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

- **protocol**: `http`
- **host**: `www.google.com`
- **port**: `80`
- **authority**: `user:password@www.google.com:80`
- **userinfo**: `user:password`
- **path**: `/command`
- **query**: `foo=bar&lorem=ipsum`
- **fragment**
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| protocol  | http                                                                 |
| host      | www.google.com                                                      |
| port      | 80                                                                  |
| authority | user:password@www.google.com:80                                    |
| userinfo  | user:password                                                      |
| path      | /command                                                           |
| query     | foo=bar&lorem=ipsum                                                |
| ref       | fragment                                                           |

URL ::= PROTOCOL
java.net.URL set(protocol, host, port, authority, userinfo, path, query, ref)

<table>
<thead>
<tr>
<th>protocol</th>
<th>http</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>port</td>
<td>80</td>
</tr>
<tr>
<td>authority</td>
<td>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80</td>
</tr>
<tr>
<td>userinfo</td>
<td>user:password</td>
</tr>
<tr>
<td>path</td>
<td>/command</td>
</tr>
<tr>
<td>query</td>
<td>foo=bar&amp;lorem=ipsum</td>
</tr>
<tr>
<td>ref</td>
<td>fragment</td>
</tr>
</tbody>
</table>

URL ::= PROTOCOL
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

param: protocol
| http
param: host
| www.google.com
param: port
| 80
param: authority
| user:password@www.google.com:80
param: userinfo
| user:password
param: path
| /command
param: query
| foo=bar&lorem=ipsum
param: ref
| fragment

URL ::= PROTOCOL '://' AUTHORITY
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

- **protocol**: `http`
- **host**: `www.google.com`
- **port**: `80`
- **authority**: `user:password@www.google.com:80`
- **userinfo**: `user:password`
- **path**: `/command`
- **query**: `foo=bar&lorem=ipsum`
- **fragment**: None

**URL** ::= **PROTOCOL** '://' **AUTHORITY**
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

param: protocol
- http

param: host
- www.google.com

param: port
- 80

param: authority
- user:password@www.google.com:80

param: userinfo
- user:password

param: path
- /command

param: query
- foo=bar&lorem=ipsum

param: ref
- fragment

URL ::= PROTOCOL '://' AUTHORITY
AUTHORITY ::= USERINFO
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

<table>
<thead>
<tr>
<th>http</th>
<th>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80/command</th>
<th>foo=bar&amp;lorem=ipsum</th>
<th>fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>param: protocol</td>
<td>http</td>
<td></td>
<td></td>
</tr>
<tr>
<td>param: host</td>
<td><a href="http://www.google.com">www.google.com</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>param: port</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>param: authority</td>
<td>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>param: userinfo</td>
<td>user:password</td>
<td></td>
<td></td>
</tr>
<tr>
<td>param: path</td>
<td>/command</td>
<td>foo=bar&amp;lorem=ipsum</td>
<td></td>
</tr>
<tr>
<td>param: query</td>
<td></td>
<td></td>
<td>fragment</td>
</tr>
<tr>
<td>param: ref</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

URL ::= PROTOCOL '://' AUTHORITY
AUTHORITY ::= USERINFO
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

<table>
<thead>
<tr>
<th>param: protocol</th>
<th>http</th>
</tr>
</thead>
<tbody>
<tr>
<td>param: host</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>param: port</td>
<td>80</td>
</tr>
<tr>
<td>param: authority</td>
<td>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80</td>
</tr>
<tr>
<td>param: userinfo</td>
<td>user:password</td>
</tr>
<tr>
<td>param: path</td>
<td>/command</td>
</tr>
<tr>
<td>param: query</td>
<td>foo=bar&amp;lorem=ipsum</td>
</tr>
<tr>
<td>param: ref</td>
<td>fragment</td>
</tr>
</tbody>
</table>

URL ::= PROTOCOL '://' AUTHORITY
AUTHORITY ::= USERINFO '@' HOST
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

<table>
<thead>
<tr>
<th>http</th>
<th>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80/command foo=bar&amp;lorem=ipsum # fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>param: protocol</td>
<td>http</td>
</tr>
<tr>
<td>param: host</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>param: port</td>
<td>80</td>
</tr>
<tr>
<td>param: authority</td>
<td>user:<a href="mailto:password@www.google.com">password@www.google.com</a>:80</td>
</tr>
<tr>
<td>param: userinfo</td>
<td>user:password</td>
</tr>
<tr>
<td>param: path</td>
<td>/command</td>
</tr>
<tr>
<td>param: query</td>
<td>foo=bar&amp;lorem=ipsum</td>
</tr>
<tr>
<td>param: ref</td>
<td>fragment</td>
</tr>
</tbody>
</table>

URL ::= PROTOCOL '://' AUTHORITY PATH '?' QUERY '#' REF
AUTHORITY ::= USERINFO '@' HOST ':' PORT
PROTOCOL ::= 'http'
USERINFO ::= 'user:password'
HOST ::= 'www.google.com'
PORT ::= '80'
PATH ::= '/command'
QUERY ::= 'foo=bar&lorem=ipsum'
REF ::= 'fragment'
URL ::= PROTOCOL '://' AUTHORITY PATH '?' QUERY '#' REF
AUTHORITY ::= USERINFO '@' HOST ':' PORT
PROTOCOL ::= 'http'
USERINFO ::= 'user:password'
HOST ::= 'www.google.com'
PORT ::= '80'
PATH ::= '/command'
QUERY ::= 'foo=bar&lorem=ipsum'
REF ::= 'fragment'

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
| http  user:password@www.google.com:80/command foo=bar&lorem=ipsum fragment
  param: protocol
  | http
  param: host
  | www.google.com
  param: port
  | 80
  param: authority
  | user:password@www.google.com:80
  param: userinfo
  | user:password
  param: path
  | /command
  param: query
  | foo=bar&lorem=ipsum
  param: ref
  | fragment
URLs

http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment
http://www.guardian.co.uk/sports/worldcup#results
ftp://bob:12345@ftp.example.com/oss/debian7.iso

URL ::= PROTOCOL '://' AUTHORITY PATH ['?' QUERY] ['#' REF]
AUTHORITY ::= [USERINFO '@'] HOST [':' PORT]
PROTOCOL ::= 'http' | 'ftp'
USERINFO ::= /[a-z]+:[a-z]+/
HOST ::= /[a-z.]+/
PORT ::= '80'
PATH ::= /\/[a-z0-9.\/]*/
QUERY ::= 'foo=bar&lorem=ipsum'
REF ::= /[a-z]+/
Implementation

\[
\begin{align*}
1 + 3 \times 6 &= 1 + 18 = 19 \\
7 \times 4 - 9 &= 28 - 9 = 19
\end{align*}
\]
Implementation

Interval Trees
Implementation

Data Flow

```
1 + 3*6
```

```
7*4 - 9
```

```
num() 1
     / 
    3   6
   /   /
num() num()
```

```
sum() 7*4-9
      /   
    7    9
   /     / 
num() num() num()
```
Implementation

Clustering

\[
\text{sum()} \quad 1 + 3 \times 6 \\
\text{num()} \\
\text{num()} \\
\text{num()} \\
\text{num()}
\]

\[
\text{sum()} \quad 7 \times 4 - 9 \\
\text{mul()} \\
\text{num()} \\
\text{num()}
\]
Implementation

Naming

\[
\text{sum()} \quad 1+3\times6 \quad \text{sum()}
\]

\[
\text{num()} \quad 1 \quad \text{mul()} \quad 3\times6 \quad \text{mul()}
\]

\[
\text{num()} \quad 3 \quad \text{num()} \quad 6
\]

\[
\text{sum()} \quad 7\times4-9 \quad \text{sum()}
\]

\[
\text{num()} \quad 7 \quad \text{num()} \quad 4 \quad \text{num()}
\]

\[
\text{num()} \quad 9
\]
Implementation

Productions

\[
\begin{align*}
1 + 3 \times 6 &= 1 + 18 \\
7 \times 4 - 9 &= 28 - 9
\end{align*}
\]
Implementation

Productions

SUM := 1+3*6 | 7*4-9
MUL := 3*6 | 7*4
NUM := 1 | 3 | 4 | 6 | 7 | 9
Implementation

Productions

SUM := NUM + MUL | MUL - NUM
MUL := NUM * NUM
NUM := 1 | 3 | 4 | 6 | 7 | 9
Implementation

Productions

Start: SUM

SUM := NUM + MUL | MUL - NUM
MUL := NUM * NUM
NUM := 1 | 3 | 4 | 6 | 7 | 9
Generalization

- **Want:** General grammar that precisely captures input format
- **Have:** Grammar that is able to reproduce the samples we learned from
- **Solution:** Generalize grammar by making assumptions on right hand sides
- **Challenge:** Assumptions might over-approximate
Active Learning

- Use test generation
- Confirm assumptions by verifying executions show predicted behaviour
- Discard refuted assumptions
- Refine confirmed assumptions
Generalizing Right Hand Sides

• Productions can be expressed as regular expressions

• Precise learning regular languages from a set of positive samples is impossible

• However: Active learning works with membership and equivalence queries

• DFA learning algorithm: L* (Angluin)
Generalizing Right Hand Sides

Assumption:

```
SUM  can be substituted by  NUM ( + | - ) NUM
```

```
sum()  1+3
  num()  1  3

sum()  7-9
  num()  7  9
```

```
sum()  num()
num()   num()
num()   num()
```
Generalizing Right Hand Sides

\[
\text{sum()} \quad \begin{array}{c}
1+3 \\
| \\
3 \\
|
\end{array} \\
\quad \begin{array}{c}
7-9 \\
| \\
7 \\
|
\end{array}
\]

\[
\text{num()} \quad \begin{array}{c}
1 \\
|
\end{array} \\
\quad \begin{array}{c}
3 \\
|
\end{array}
\]

\[
\text{num()} \quad \begin{array}{c}
7 \\
|
\end{array} \\
\quad \begin{array}{c}
9 \\
|
\end{array}
\]

Start: \textbf{SUM}

\[
\begin{align*}
\text{SUM} & : = \ ( \text{MUL} \ \text{NUM} \ ) \ ( \ + \ | \ - \ ) ( \ \text{MUL} \ \text{NUM} \ ) \\
\text{MUL} & : = \ \text{NUM} \ \ast \ \text{NUM} \\
\text{NUM} & : = \ 1 \ | \ 3 \ | \ 4 \ | \ 6 \ | \ 7 \ | \ 9
\end{align*}
\]
Generalizing Terminal Symbols

\[
\text{NUM} := 1 \mid 3 \mid 4 \mid 6 \mid 7 \mid 9
\]
Generalizing Terminal Symbols

- High probability of unobserved characters
- Large alphabets result in large performance hits for $L^*$
- **Approach:** Regular Expression Hierarchy
Regex Hierarchy

- [0-9a-zA-Z]*
- [-][1-9][0-9]*
- [0-9]*
- [1-9][0-9]*/
- [0-9]
Generalizing Terminal Symbols

\[
\text{NUM} := [0 \ -\ 9]
\]
Generalized Grammar

Start: **SUM**

**SUM** := ( **MUL** | **NUM** )( + | - )( **MUL** | **NUM** )

**MUL** := **NUM** * **NUM**

**NUM** := [ 0 - 9 ]
INI Files

[Application]
Version = 0.5
WorkingDir = /tmp/mydir/
[User]
User = Bob
Password = 12345

INI ::= LINE+
LINE ::= SECTION_LINE \r
          | OPTION_LINE \r
SECTION_LINE ::= [' \nKEY ']'
OPTION_LINE ::= KEY ' = ' VALUE
KEY ::= /[a-zA-Z]*/
VALUE ::= /[a-zA-Z0-9\]/
JSON ::= WHITESPACE VALUE WHITESPACE
WHITESPACE ::= /[ \n\t]+/
VALUE ::= JSONOBJECT | ARRAY | STRING |
    TRUE | FALSE | NULL | NUMBER
TRUE ::= 'true'
FALSE ::= 'false'
NULL ::= 'null'
NUMBER ::= ['-'] /[0-9]+/
STRING ::= """ INTERNALSTRING """
INTERNALSTRING ::= /[^a-zA-Z0-9 ]+/ARRAY ::=
    '[
        [WHITESPACE]
        [VALUE [WHITESPACE]
        [',', [WHITESPACE] VALUE [WHITESPACE]]+
    ]'
JSONOBJECT ::=
    '{
        [WHITESPACE]
        [STRING [WHITESPACE] ':': [WHITESPACE]
        VALUE [WHITESPACE]
        [',', [WHITESPACE]
        STRING [WHITESPACE] ':': [WHITESPACE]
        VALUE [WHITESPACE]]+
    }'
# Evaluation

<table>
<thead>
<tr>
<th>Subject</th>
<th>Accuracy</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.Url</td>
<td>82.3 %</td>
<td>100 %</td>
</tr>
<tr>
<td>Apache Commons CSV</td>
<td>100 %</td>
<td>100 %</td>
</tr>
<tr>
<td>java.util.Properties</td>
<td>100 %</td>
<td>100 %</td>
</tr>
<tr>
<td>INI4J</td>
<td>64.6 %</td>
<td>100 %</td>
</tr>
<tr>
<td>Minimal JSON</td>
<td>100 %</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Hörschele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Challenges

- Table based parsers
- Binary formats
- Context sensitive features
- Multi-layer input formats
- Learning grammars without samples
Program Behavior

Input(s) → Program
Program Behavior

Input(s) \rightarrow Program

xyzzy
Program Behavior

Input(s) → Program

xyzzy

✘
Program Behavior

- checks for digit
Program Behavior

Input(s) → Program

- checks for digit
- checks for "true"/"false"
Program Behavior

Input(s) ➔ Program

xyzzy

- checks for digit
- checks for "true"/"false"
- checks for ""

✘
Program Behavior

- checks for digit
- checks for "true"/"false"
- checks for ""
- checks for '['

Input(s) ➔ Program

xyzzy
Program Behavior

- checks for digit
- checks for "true"/"false"
- checks for '"'
- checks for '['
- checks for '{'
Program Behavior

- checks for digit
- checks for "true"/'false"
- checks for ""
- checks for '['
- checks for '{'
Program Behavior

Input → Program
Program Behavior

Input → Program

0
Program Behavior

Input

0

Program

✔️
Program Behavior

Input

0

Program

✔
Program Behavior

- checks for digit
- checks for "true"/"false"
- checks for ""
- checks for '['
- checks for '{'

Input → Program

0 ✔
Program Behavior

- checks for digit
- checks for "true"/"false"
- checks for ""
- checks for '['
- checks for '{'
Program Behavior

- checks for digit
- checks for "true"/"false"
- checks for ""
- checks for '['
- checks for '{'
Program Behavior

Input ➔ Program

"" ➔ "" ✗
Program Behavior

Input ➔ Program

"" ➔ X
Program Behavior

- checks for ""
- checks for '\'
- checks for character
Program Behavior

- checks for ""
- checks for '\'
- checks for character
Program Behavior

Input ➔ Program

...✔
JSON Files (no samples)

<JSON files>

```
JSON ::= VALUE
VALUE ::= OBJECT | ARRAY | STRING |
       TRUE | FALSE | NULL | NUMBER
OBJECT ::= '{' [NAME ':' VALUE [ ']' ] ] '}'
ARRAY ::= '[' [VALUE [',' VALUE [ ']' ] ] +'] ']
STRING ::= """" | """" ENDCAPTURE [ '"' ]
ENDCAPTURE ::= ('e' | 'E' | 's' | '0' | '"' )+
TRUE ::= 'true'
FALSE ::= 'false'
NULL ::= 'null'
NAME ::= """
NUMBER ::= /[0-9]+/[ FRACTION | EXPONENT ]
FRACTION ::= ".0"
EXPONENT ::= ('e' | 'E' | '+' | '-' | '0' )+
```
Learned Grammars
Learned Grammars

- give insights into the structure of inputs
  - reverse engineering
  - writing tests
  - writing parsers
Learned Grammars

• give insights into the structure of inputs
  → reverse engineering
  → writing tests
  → writing parsers

• can directly be used for test generators
  → automated fuzzing tools
Mining Input Grammars

https://www.st.cs.uni-saarland.de/models/autogram/