Grammars for Mutation Testing

Andreas Zeller
CISPA Helmholtz Center for Information Security

www.fuzzingbook.org

@AndreasZeller

Saarbrücken

CISPA

HELMHOLTZ-ZENTRUM i. G.

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

SAARLAND

Scientific excellence in fundamental research

50,000,000 €/year • 500+ researchers

Fuzzing

Random Testing at the System Level

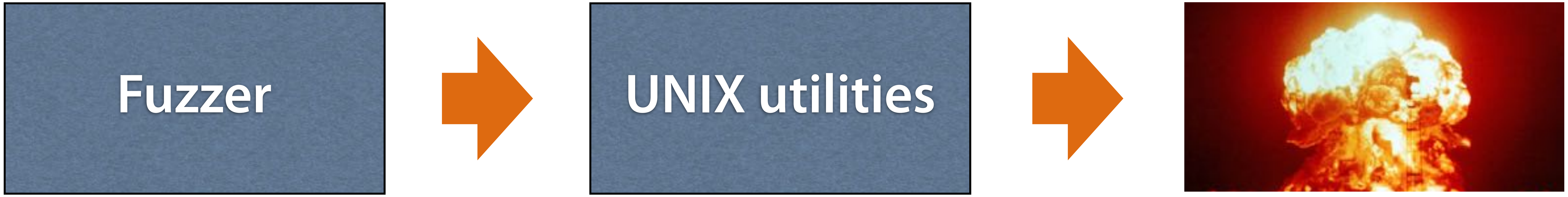
```
[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!AxB"YXRS@!
Kd6;wtAMefFWM(`|J_<1~o}z3K(CCzRH JIIvHz>_*.\>JrlU32~eGP?
lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-i,c{<[~m!]o;{.'}Gj\(X}
EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6}0|
Ko=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`+t*gka<W=Z.
%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?[_[Z^LBMPG-
FKj'\xwuZ1=Q`^`5,$N$Q@[!CuRzJ2D|vBy!^zkhdf3C5PAkR?V hn|
3='i2Qx]D$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%|b`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r
```

Bart Miller

University of Wisconsin-Madison

Fuzzing

Random Testing at the System Level



Fuzzer

UNIX utilities

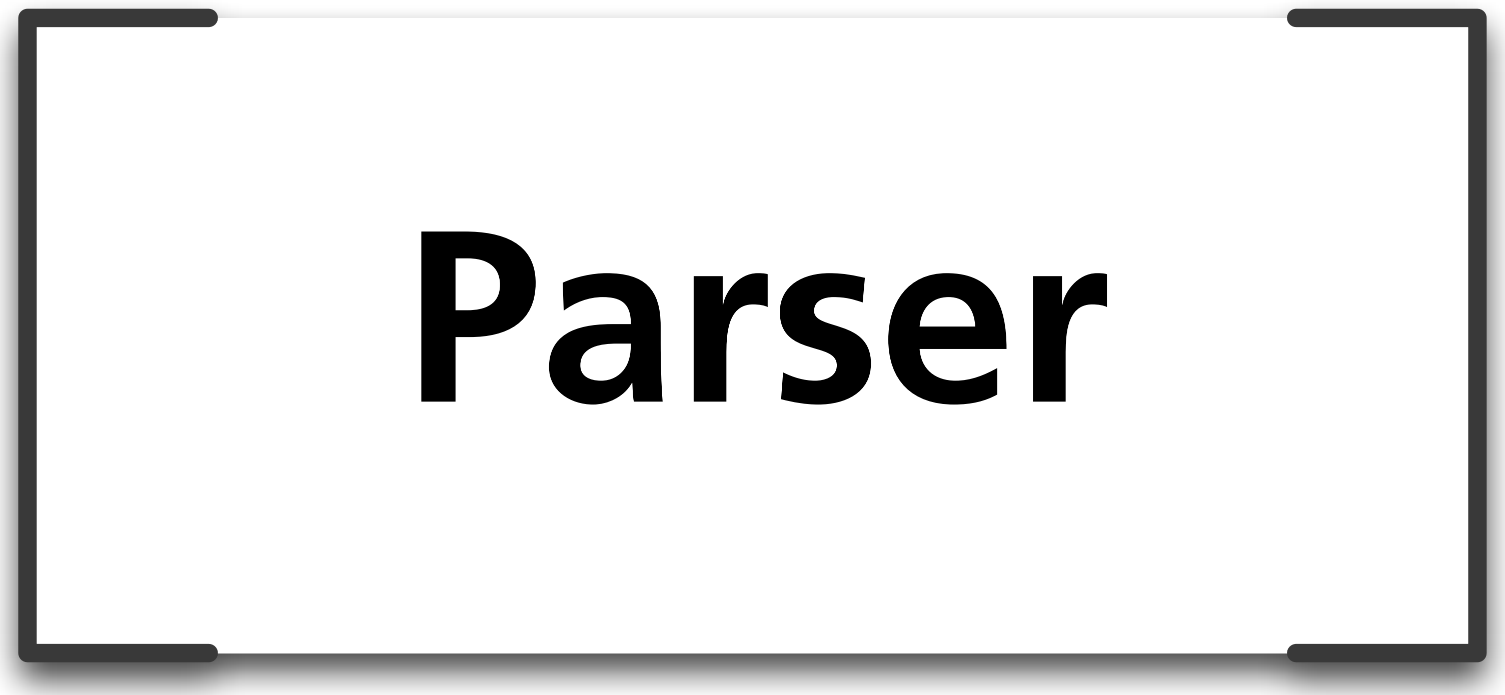
“ab’d&gfdfggg”

grep • sh • sed …

25%–33%

Grammar Fuzzing

- Suppose you want to test a *parser* –
  to compile and execute a program
- To get deep into the program, you need *syntactically correct inputs*



Parser

LangFuzz (2012)

- Fuzz tester for JavaScript and other languages
- Uses a full-fledged *grammar* to generate inputs

JavaScript Grammar

If Statement

IfStatement^{full} ⇒
  `if` *ParenthesizedExpression* *Statement*^{full}
| `if` *ParenthesizedExpression* *Statement*^{noShortIf} `else` *Statement*^{full}
IfStatement^{noShortIf} ⇒ `if` *ParenthesizedExpression* *Statement*^{noShortIf} `else` *Statement*^{noShortIf}

Switch Statement

SwitchStatement ⇒
  `switch` *ParenthesizedExpression* `{ }`
| `switch` *ParenthesizedExpression* `{` *CaseGroups* *LastCaseGroup* `}`
CaseGroups ⇒
  «empty»
| *CaseGroups* *CaseGroup*
CaseGroup ⇒ *CaseGuards* *BlockStatementsPrefix*

Fuzzing JavaScript

```
var haystack = "foo";
var re_text  = "^foo";
haystack += "x";
re_text  += "(x)";
var re = new RegExp(re_text);
re. test(haystack);
Reg
prin
```

JavaScript Parser

30 Chromium + Mozilla Security Rewards
53,000 US$ in Bug Bounties

C. Holler

Holler, Herzig, Zeller: "Fuzzing with Code Fragments", USENIX 2012

LangFuzz (2012)

- Fuzz tester for JavaScript and other languages
- Uses a full-fledged *grammar* to generate inputs
- **Uses grammar
to *parse and mutate existing inputs***

Mutating with Grammars

To use a grammar for mutating code and data,

1. **Parse** an input into a derivation tree
2. **Mutate** the derivation tree
3. **Write** the tree out again

You need a *grammar*, a *parser*, and an *unparser*

A Grammar Framework in Python

We have implemented a full-fledged *Python framework* to

1. **Specify** grammars
2. **Parse** inputs into derivation trees
3. **Mutate** derivation trees
4. **Write** the tree out again

plus much much more; e.g. testing

This framework comes implemented as *Jupyter notebooks*

A Grammar

```
<start>   ::= <expr>
<expr>    ::= <term> + <expr> | <term> - <expr> | <term>
<term>    ::= <term> * <factor> | <term> / <factor> | <factor>
<factor>  ::= +<factor> | -<factor> | (<expr>)
              | <integer> | <integer>.<integer>
<integer> ::= <digit><integer> | <digit>
<digit>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

A Grammar in Python

```
EXPR_GRAMMAR = {
    "<start>":  ["<expr>"],
    "<expr>":   ["<term> + <expr>", "<term> - <expr>", "<term>"],
    "<term>":   ["<factor> * <term>", "<factor> / <term>", "<factor>"],
    "<factor>": ["+<factor>", "-<factor>", "(<expr>)",
                 "<integer>.<integer>", "<integer>"],
   "<integer>": ["<digit><integer>", "<digit>"],
    "<digit>":  ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
```

Parsing with Grammars

```
expr_input = "2 + -2"
expr_parser = EarleyParser(EXPR_GRAMMAR)
expr_trees = expr_parser.parse(expr_input)
```

Mutating with Grammars

```
def swap_plus_minus(tree):
    node, children = tree
    if node == " + ":
        node = " - "
    elif node == " - ":
        node = " + "
    return node, children

def apply_mutator(tree, mutator):
    node, children = mutator(tree)
    return node, [apply_mutator(c, mutator) for c in children]

mutated_tree = apply_mutator(expr_tree, swap_plus_minus)
```

Demo

Jupyter Notebooks

- **Very** fast prototyping
- Literate programming with examples (and tests!)
- Data visualizations

Prototype for Python first; then go for a "serious" language like C

Learning Grammars

```
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]+/
```

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

Parser-Directed Fuzzing

We track and satisfy *comparisons* in parsers to find language elements

`{ [ ( + & ?` *identifier number* …
`+= == ++ /= &= |= != if in` *string* …
`=== !== <<= >>> for try let` …
`>>>= true null void with else` …
`false throw while break catch` …
`return delete typeof Object` …
`default finally indexOf`
`continue function debugger`
`undefined stringify`
`instanceof`

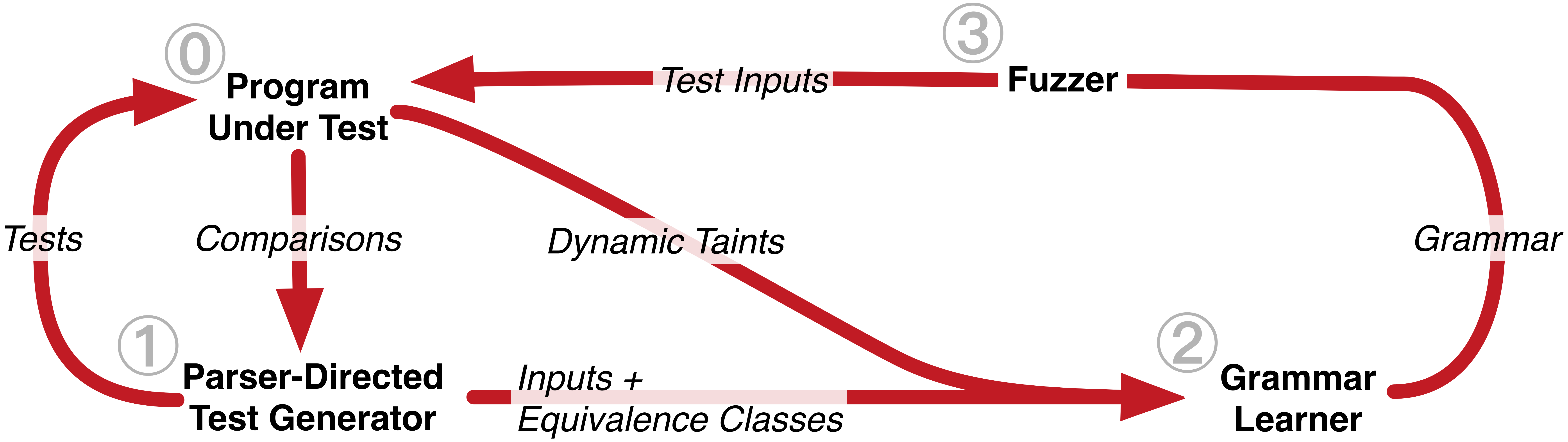
JS tokens of length 3+ discovered

AFL	**KLEE**	**pFuzzer**
5,0 %	7,5 %	52,5 %

Mathis, Gopinath, Mera, Kampmann, Höschele, Zeller: *Parser-Directed Fuzzing*, PLDI 2019

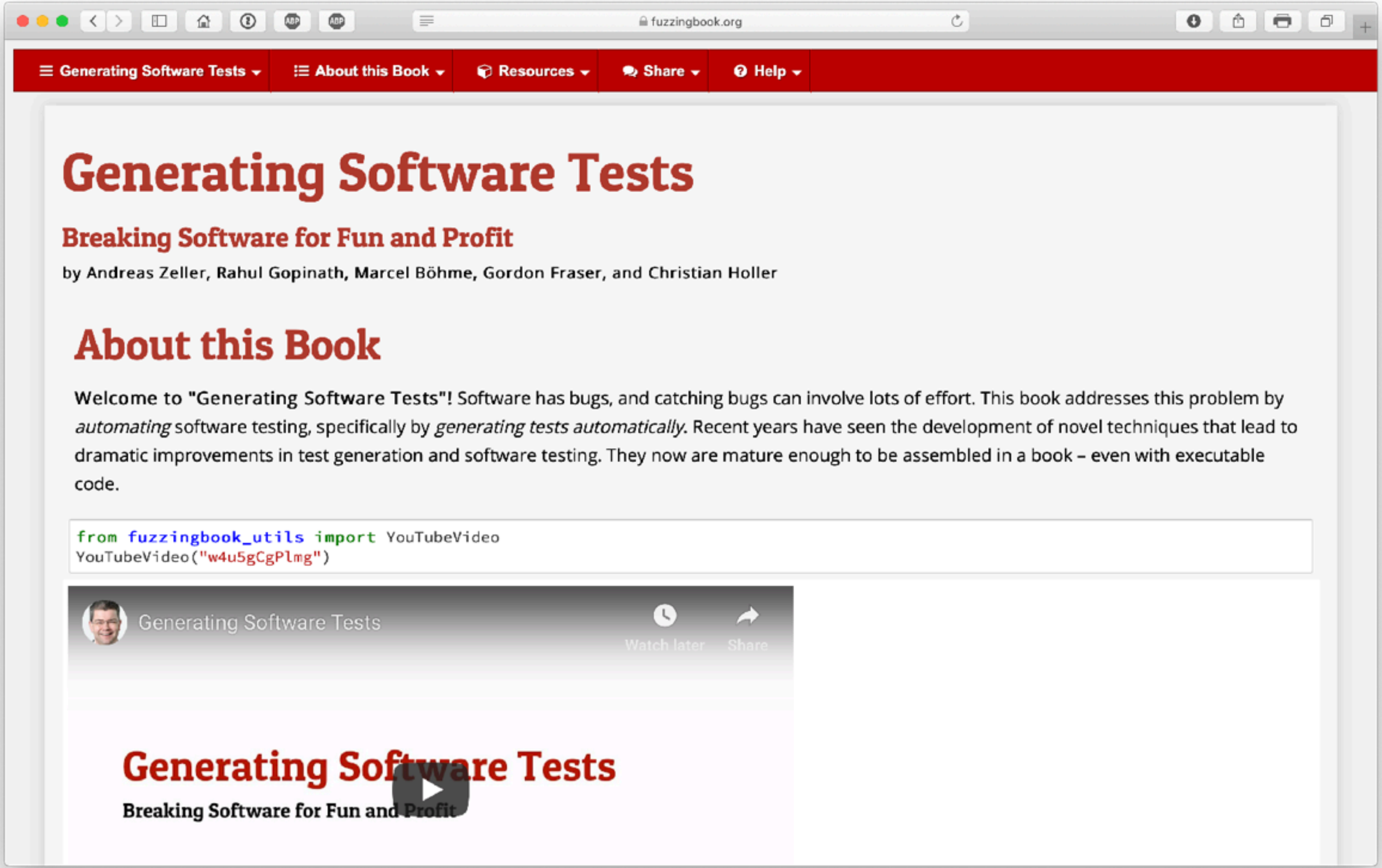
Deep Fuzzing without Samples

PYGMALION prototype for Python programs



Perfect coverage, much faster than AFL, much better structure than KLEE

Gopinath, Mathis, Höschele, Kampmann, Zeller: "Sample-Free Learning of Input Grammars"



Generating Software Tests

Breaking Software for Fun and Profit

by Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler

About this Book

Welcome to "Generating Software Tests"! Software has bugs, and catching bugs can involve lots of effort. This book addresses this problem by *automating* software testing, specifically by *generating tests automatically*. Recent years have seen the development of novel techniques that lead to dramatic improvements in test generation and software testing. They now are mature enough to be assembled in a book – even with executable code.

```
from fuzzingbook_utils import YouTubeVideo
YouTubeVideo("w4u5gCgPlmg")
```

Generating Software Tests
Watch later
Share
Generating Software Tests
Breaking Software for Fun and Profit

A Grammar Framework in Python

We have implemented a full-fledged *Python framework* to

1. **Specify** grammars
2. **Parse** inputs into derivation trees
3. **Mutate** derivation trees
4. **Write** the tree out again

plus much much more;
e.g. testing

This framework comes implemented as *Jupyter notebooks*

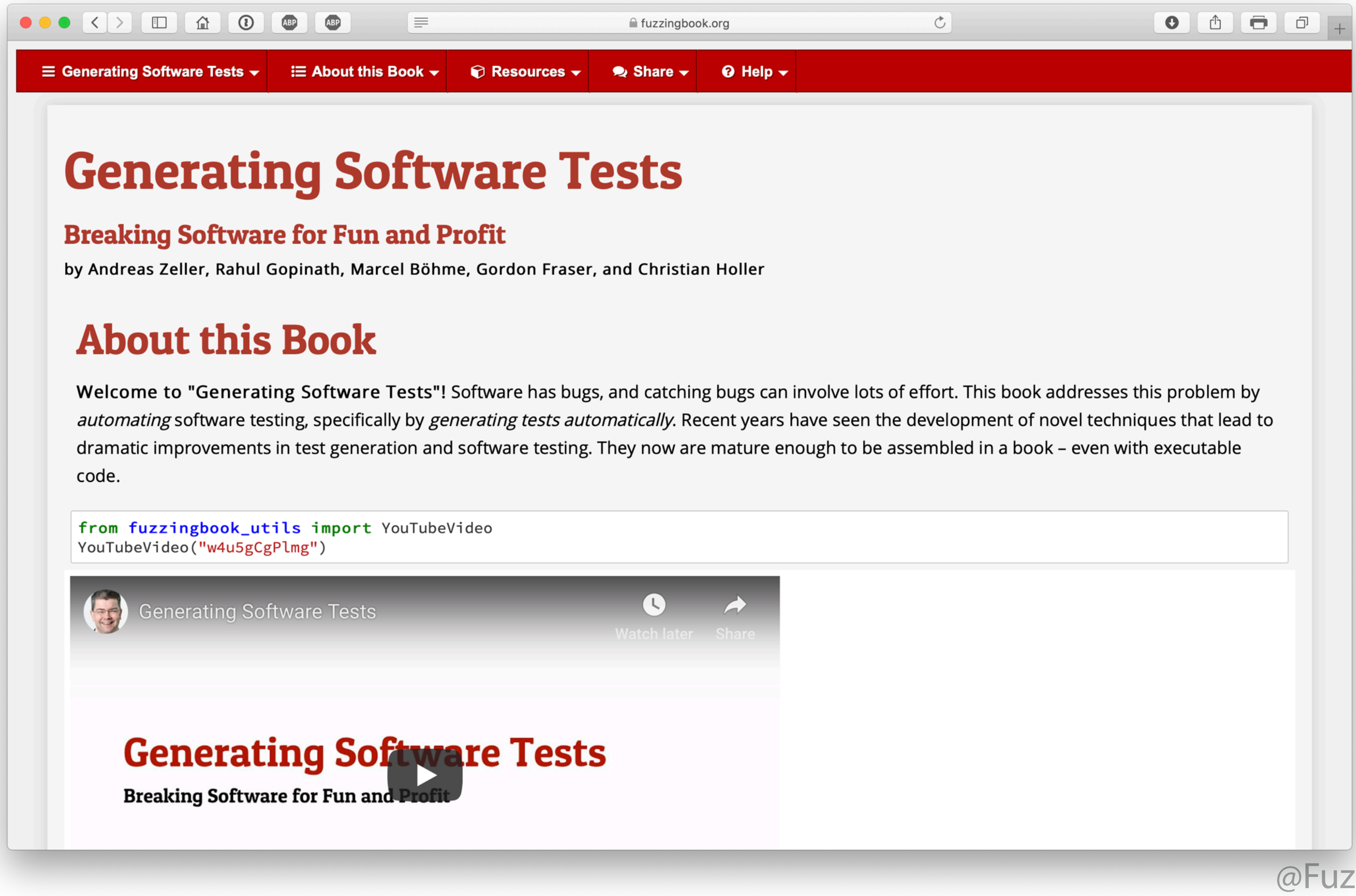
A Grammar in Python

```
EXPR_GRAMMAR = {
    "<start>":  ["<expr>"],
    "<expr>":   ["<term> + <expr>", "<term> - <expr>", "<term>"],
    "<term>":   ["<factor> * <term>", "<factor> / <term>", "<factor>"],
    "<factor>": ["+<factor>", "-<factor>", "(<expr>)",
                 "<integer>.<integer>", "<integer>"],
   "<integer>": ["<digit><integer>", "<digit>"],
    "<digit>":  ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
```

Jupyter Notebooks

- **Very** fast prototyping
- Literate programming with examples (and tests!)
- Data visualizations

Prototype for Python first; then go for a "serious" language like C



Generating Software Tests

Breaking Software for Fun and Profit

by Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler

About this Book

Welcome to "Generating Software Tests"! Software has bugs, and catching bugs can involve lots of effort. This book addresses this problem by *automating* software testing, specifically by *generating tests automatically*. Recent years have seen the development of novel techniques that lead to dramatic improvements in test generation and software testing. They now are mature enough to be assembled in a book – even with executable code.

```
from fuzzingbook_utils import YouTubeVideo
YouTubeVideo("w4u5gCgPlmg")
```

Generating Software Tests
Watch later
Share
Generating Software Tests
Breaking Software for Fun and Profit

www.fuzzingbook.org

Mutations with Grammars

A Chapter of "Generating Software Tests"

Author:
Andreas Zeller,
Rahul Gopinath,
Marcel Böhme,
Gordon Fraser, and
Christian Holler

April 22, 2019

Contents

1 Mutations with Grammars **3**

1.1 Defining Grammars 3

1.2 Fuzzing with Grammars 5

1.3 Parsing with Grammars 6

1.4 Mutating a Tree 8

1.5 Unparsing the Mutated Tree 9

1.6 Lots of mutations 9

1.7 Another Example: JSON 10

2 References **21**

List of Figures

List of Tables

List of Codes

A chapter of **Generating Software Tests**, by Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler.

1 Mutations with Grammars

In this notebook, we make a very short and simple introduction on how to use the `fuzzingbook` framework for grammar-based mutation – both for data and for code.

Prerequisites

- This chapter is meant to be self-contained.

1.1 Defining Grammars

We define a grammar using standard Python data structures. Suppose we want to encode this grammar:

```
<start>   ::= <expr>
<expr>    ::= <term> + <expr> | <term> - <expr> | <term>
<term>    ::= <term> * <factor> | <term> / <factor> | <factor>
<factor>  ::= +<factor> | -<factor> | (<expr>) | <integer> | <integer>.<integer>
<integer> ::= <digit><integer> | <digit>
<digit>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```python
import fuzzingbook_utils
```

```python
from Grammars import syntax_diagram, is_valid_grammar,
    ↪ convert_ebnf_grammar, srange, crange
```

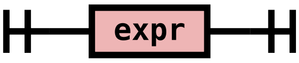
In Python, we encode this as a mapping (a dictionary) from nonterminal symbols to a list of possible expansions:

```python
EXPR_GRAMMAR = {
    ”<start>”:
        [”<expr>”],

    ”<expr>”:
        [”<term> + <expr>”, ”<term> - <expr>”, ”<term>”],

    ”<term>”:
        [”<factor> * <term>”, ”<factor> / <term>”, ”<factor>”],

    ”<factor>”:
        [”+<factor>”,
         ”-<factor>”,
         ”(<expr>)”,
         ”<integer>.<integer>”,
         ”<integer>”],

    ”<integer>”:
        [”<digit><integer>”, ”<digit>”],

    ”<digit>”:
        [”0”, ”1”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”]
}
```

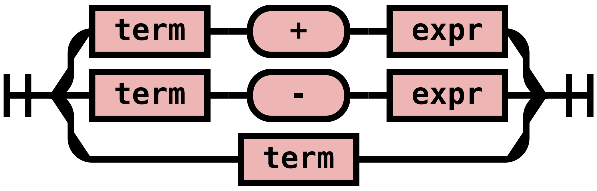
```
assert is_valid_grammar(EXPR_GRAMMAR)
```

```
syntax_diagram(EXPR_GRAMMAR)
```

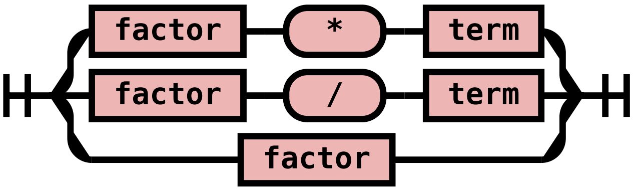
```
start
```



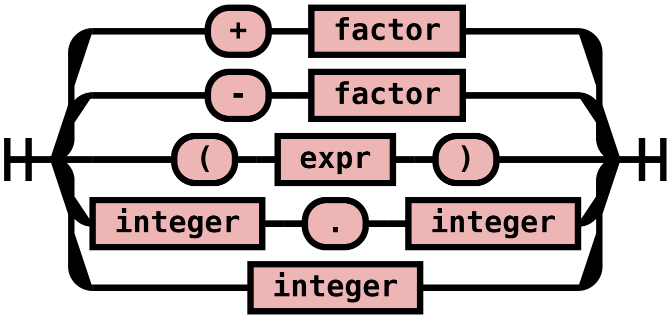
```
expr
```



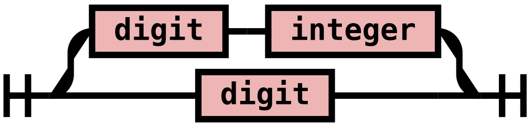
```
term
```



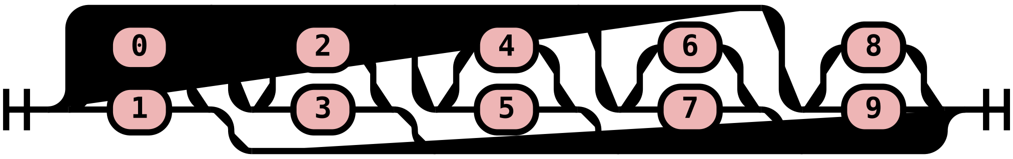
```
factor
```



```
integer
```



```
digit
```



1.2 Fuzzing with Grammars

We mostly use grammars for *fuzzing*, as in here:

```python
from GrammarFuzzer import GrammarFuzzer
```

```python
expr_fuzzer = GrammarFuzzer(EXPR_GRAMMAR)
for i in range(10):
    print(expr_fuzzer.fuzz())
```

```
3.8 + --62.912 - ++4 - +5 * 3.0 * 4
7 * (75.5 - -6 + 5 - 4) + -(8 - 1) / 5 * 2
(-(9) * +6 + 9 / 3 * 8 - 9 * 8 / 7) / -+-65
(9 + 8) * 2 * (6 + 6 + 9) * 0 * 1.9 * 0
(1 * 7 - 9 + 5) * 5 / 0 * 5 + 7 * 5 * 7
```

```
-(6 / 9 - 5 - 3 - 1) - -1 / +1 + (9) / (8) * 6
(+-(0 - (1) * 7 / 3)) / ((1 * 3 + 8) + 9 - +1 / --0) - 5 *
(-+939.491)
+2.9 * 0 / 501.19814 / --+--(6.05002)
+-8.8 / (1) * -+1 + -8 + 9 - 3 / 8 * 6 + 4 * 3 * 5
(+(8 / 9 - 1 - 7)) + ---06.30 / +4.39
```

1.3 Parsing with Grammars

We can parse a given input using a grammar:

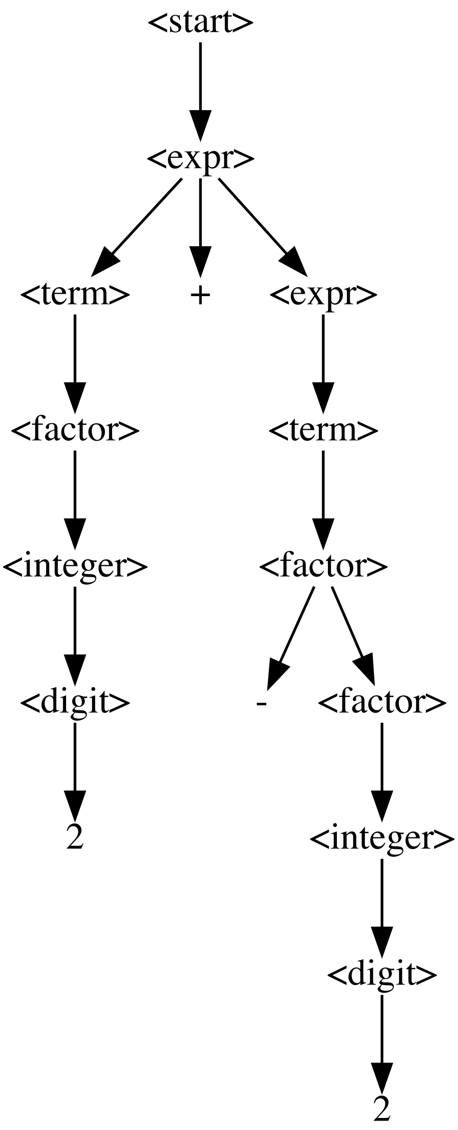
```
expr_input = ”2 + -2”
```

```
from Parser import EarleyParser, display_tree, tree_to_string
```

```
expr_parser = EarleyParser(EXPR_GRAMMAR)
```

```
expr_tree = list(expr_parser.parse(expr_input))[0]
```

```
display_tree(expr_tree)
```



Internally, each subtree is a pair of a node and a list of children (subtrees)

```
expr_tree
```

```
('<start>',
 [('<expr>',
   [('<term>', [('<factor>', [('<integer>', [('<digit>', [('2', [])
])])])]),
    (' + ', []),
    ('<expr>',
     [('<term>',
       [('<factor>',
         [('-', []),
          ('<factor>', [('<integer>', [('<digit>', [('2', [])])])])
])])])])])
```

1.4 Mutating a Tree

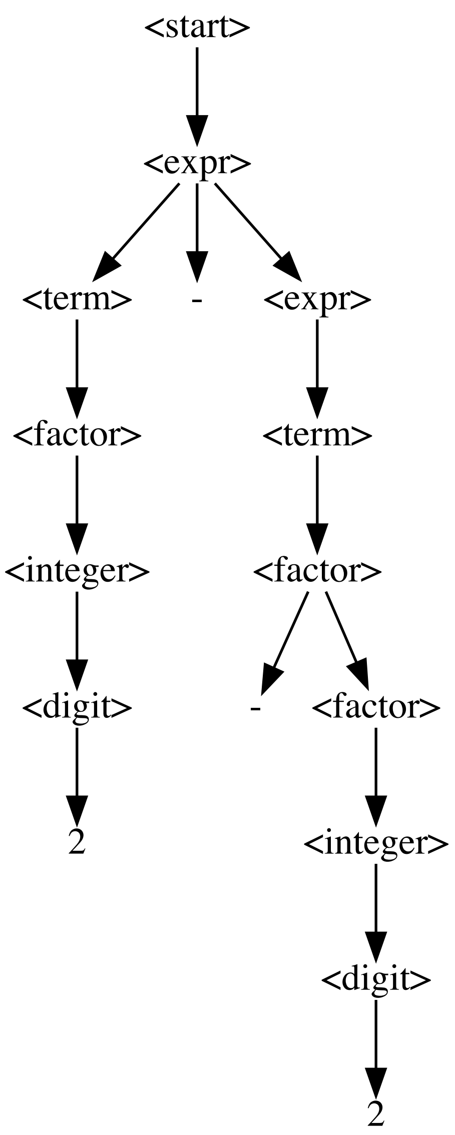
We define a simple mutator that traverses an AST to mutate it.

```
def swap_plus_minus(tree):
    node, children = tree
    if node == ” + ”:
        node = ” – ”
    elif node == ” – ”:
        node = ” + ”
    return node, children
```

```
def apply_mutator(tree, mutator):
    node, children = mutator(tree)
    return node, [apply_mutator(c, mutator) for c in children]
```

```
mutated_tree = apply_mutator(expr_tree, swap_plus_minus)
```

```
display_tree(mutated_tree)
```



1.5 Unparsing the Mutated Tree

To unparse, we traverse the tree and look at all terminal symbols:

```
tree_to_string(mutated_tree)
```

```
'2 - -2'
```

1.6 Lots of mutations

```
for i in range(10):
    s = expr_fuzzer.fuzz()
    s_tree = list(expr_parser.parse(s))[0]
    s_mutated_tree = apply_mutator(s_tree, swap_plus_minus)
    s_mutated = tree_to_string(s_mutated_tree)
    print('    ' + s + '\n->  ' + s_mutated + '\n')
```

```
    8786.82 - +01.170 / 9.2 - +(7) + 1 * 9 - 0
->  8786.82 + +01.170 / 9.2 + +(7) - 1 * 9 + 0

    +-6 * 0 / 5 * (-(1.7 * +(-1 / +4.9 * 5 * 1 * 2) + -4.2 + (6 +
-5) / (4 * 3 + 4)))
->  +-6 * 0 / 5 * (-(1.7 * +(-1 / +4.9 * 5 * 1 * 2) - -4.2 - (6 -
-5) / (4 * 3 - 4)))

    (6 * 2 + 5) * -(5) / (0 + 7) / 7 - -075 / 2
->  (6 * 2 - 5) * -(5) / (0 - 7) / 7 + -075 / 2

    6 + 9 * 3 * 7 - 6 / 0 * 5 - 7 * 5 + 3 - 0
->  6 - 9 * 3 * 7 + 6 / 0 * 5 + 7 * 5 - 3 + 0

    93 * +-(0 / 0 - 0 - 4) / (2) / 1 - 2.49 - (7.0 / 9.1)
->  93 * +-(0 / 0 + 0 + 4) / (2) / 1 + 2.49 + (7.0 / 9.1)

    +0.6 * 1.62 * 3 / 7 * 5 - 645 / (3 * 4 - 2) / 7
->  +0.6 * 1.62 * 3 / 7 * 5 + 645 / (3 * 4 + 2) / 7

    (1 * 8 * 4 + 1) - +-+(2 - 8) / 0.76 * 3
->  (1 * 8 * 4 - 1) + +-+(2 + 8) / 0.76 * 3

    -+-+-(0 - 0) / 1 / 3 / 5 * 9 * 2 + +5.0 / (+(5) * 8 * 7)
->  -+-+-(0 + 0) / 1 / 3 / 5 * 9 * 2 - +5.0 / (+(5) * 8 * 7)

    1 * ++6 - -(5 + 7 + 5 - 6 - 4) - 5.4 / 2 - +5 / 9
->  1 * ++6 + -(5 - 7 - 5 + 6 + 4) + 5.4 / 2 + +5 / 9

    (1.5 * 1 + 9 - 3 + 3) - 6 / 6 + 1 + 0
->  (1.5 * 1 - 9 + 3 - 3) + 6 / 6 - 1 - 0
```

1.7 Another Example: JSON

```
import string
```

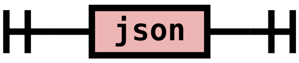
```
CHARACTERS_WITHOUT_QUOTE = (string.digits
                             + string.ascii_letters
                             + string.punctuation.replace('”', '').replace('\\', '')
                             + ' ')
```

```
JSON_EBNF_GRAMMAR = {
    ”<start>”: [”<json>”],
    ”<json>”: [”<element>”],
    ”<element>”: [”<ws><value><ws>”],
    ”<value>”: [”<object>”, ”<array>”, ”<string>”, ”<number>”, ”true”, ”false”, ”null”],
    ”<object>”: [”{<ws>}”, ”{<members>}”],
    ”<members>”: [”<member>(,<members>)*”],
    ”<member>”: [”<ws><string><ws>:<element>”],
    ”<array>”: [”[<ws>]”, ”[<elements>]”],
    ”<elements>”: [”<element>(,<elements>)*”],
    ”<element>”: [”<ws><value><ws>”],
    ”<string>”: ['”' + ”<characters>” + '”'],
    ”<characters>”: [”<character>*”],
    ”<character>”: srange(CHARACTERS_WITHOUT_QUOTE),
    ”<number>”: [”<int><frac><exp>”],
    ”<int>”: [”<digit>”, ”<onenine><digits>”, ”-<digits>”, ”-<onenine><digits>”],
    ”<digits>”: [”<digit>+”],
    ”<digit>”: ['0', ”<onenine>”],
    ”<onenine>”: crange('1', '9'),
    ”<frac>”: [””, ”.<digits>”],
    ”<exp>”: [””, ”E<sign><digits>”, ”e<sign><digits>”],
    ”<sign>”: [””, '+', '-'],
    ”<ws>”: [”( )*”]
}

assert is_valid_grammar(JSON_EBNF_GRAMMAR)
```

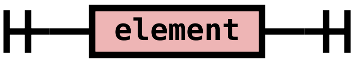
```
JSON_GRAMMAR = convert_ebnf_grammar(JSON_EBNF_GRAMMAR)
```

```
syntax_diagram(JSON_GRAMMAR)
```

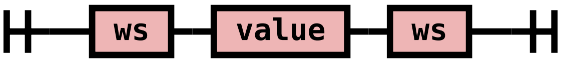
```
start
```



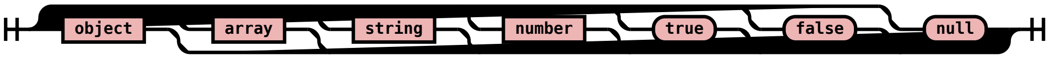
```
json
```



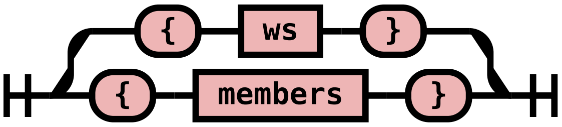
```
element
```



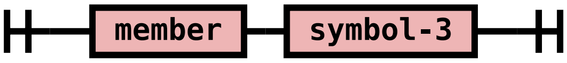
```
value
```



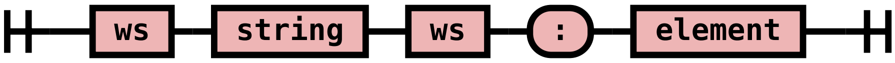
```
object
```



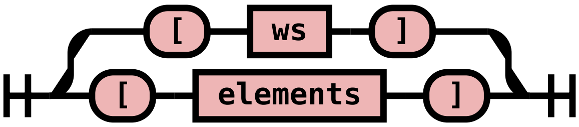
```
members
```



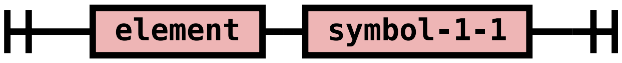
```
member
```



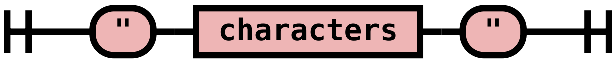
```
array
```



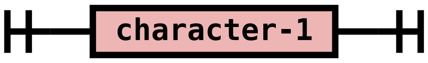
```
elements
```



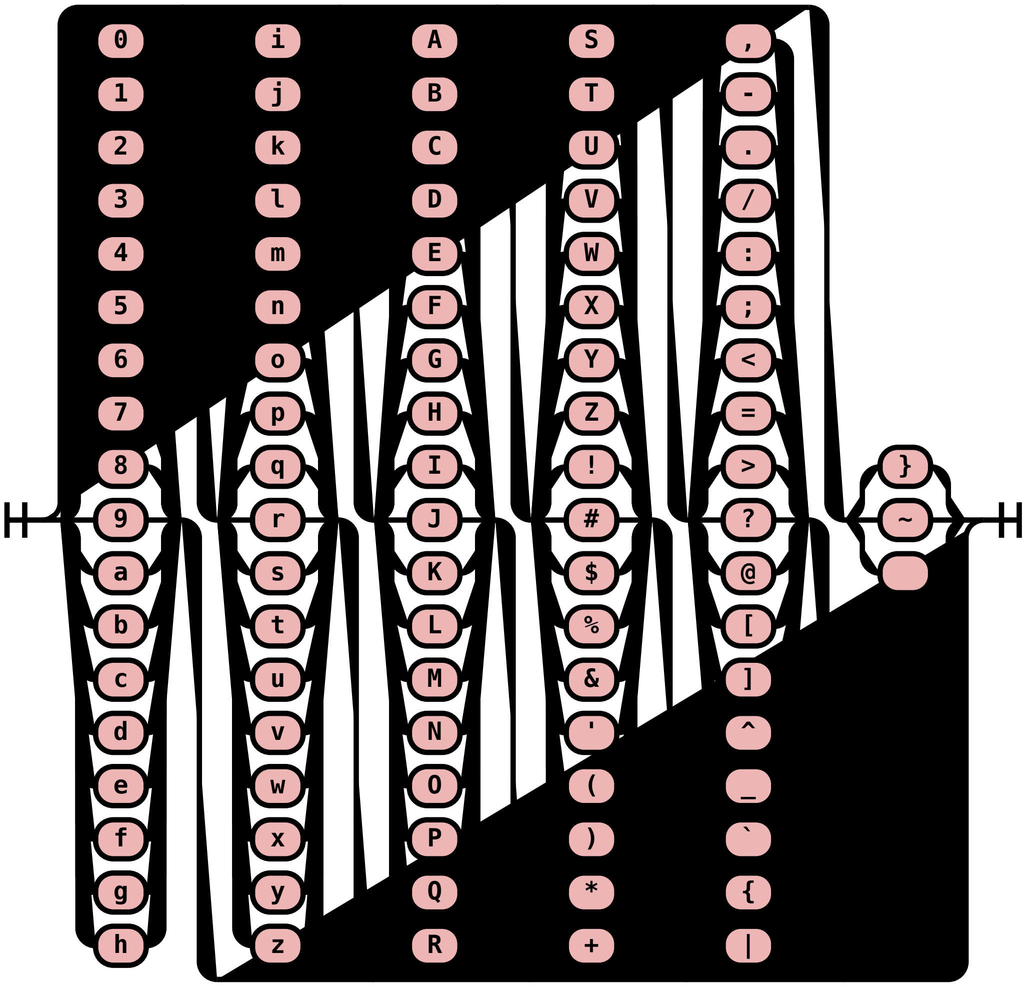
```
string
```



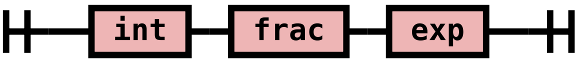
```
characters
```



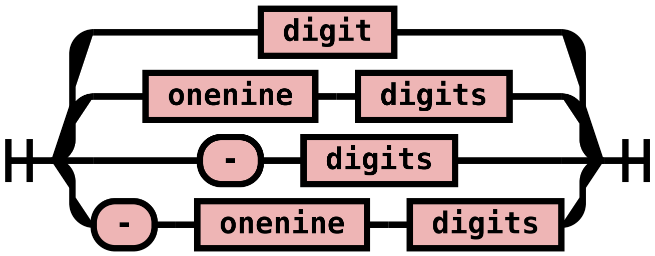
```
character
```


0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
!
#
$
%
&
'
(
)
*
+
,
-
.
/
:
;
<
=
>
?
@
[
]
^
_
`
{
|
}
~

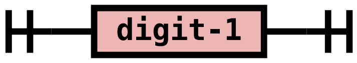
```
number
```


int
frac
exp

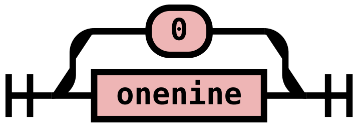
```
int
```



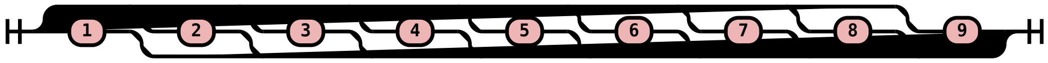
digits



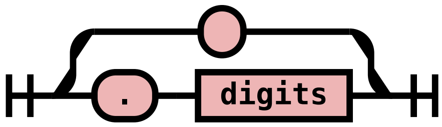
digit



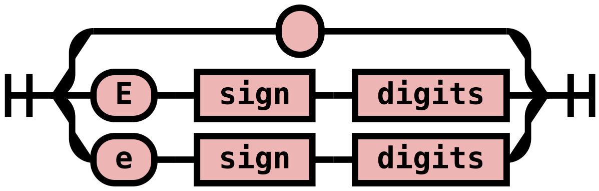
onenine



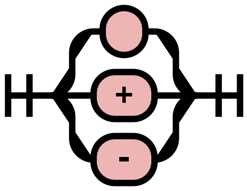
frac



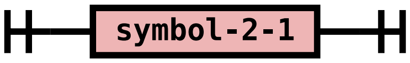
```
exp
```



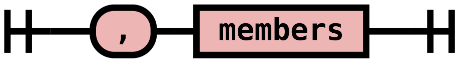
```
sign
```



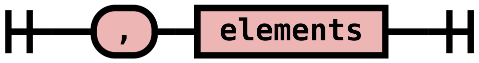
```
ws
```



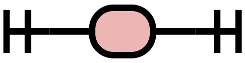
```
symbol
```



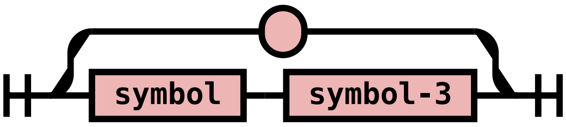
```
symbol-1
```



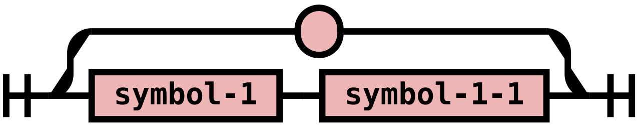
```
symbol-2
```



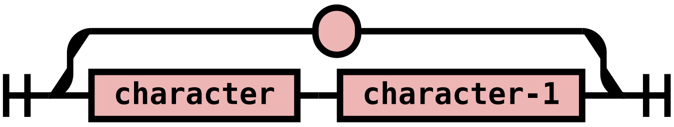
```
symbol-3
```



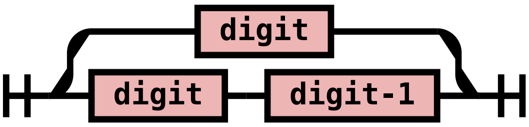
```
symbol-1-1
```



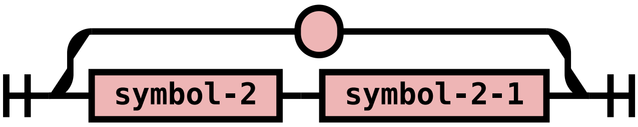
```
character-1
```



```
digit-1
```



```
symbol-2-1
```

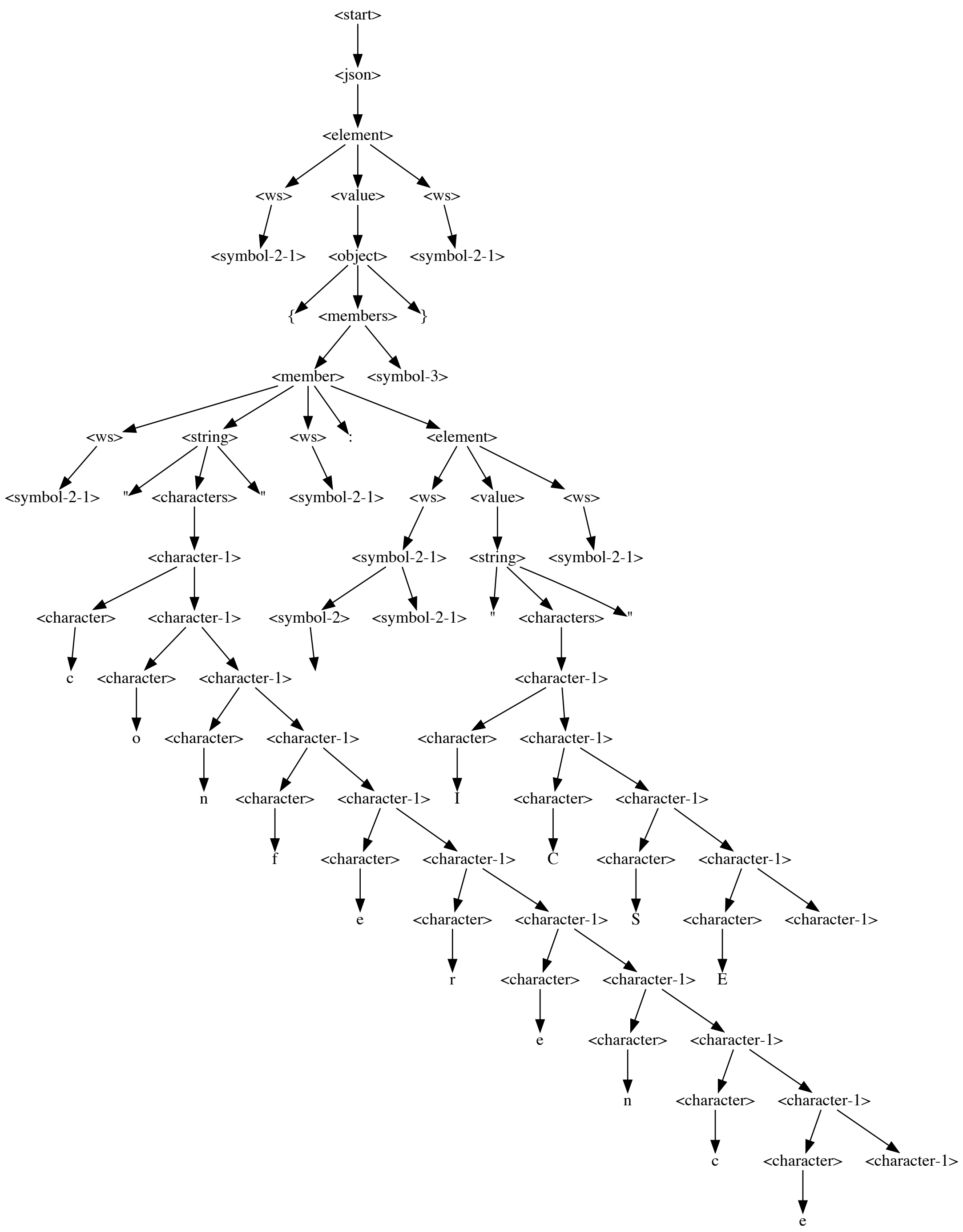


```python
json_input = '{"conference": "ICSE"}'
```

```python
json_parser = EarleyParser(JSON_GRAMMAR)
```

```python
json_tree = list(json_parser.parse(json_input))[0]
```

```python
display_tree(json_tree)
```



```python
def swap_venue(tree):
    if tree_to_string(tree) == '”ICSE”':
        tree = list(json_parser.parse('”ICST”'))[0]
    return tree
```

```python
mutated_tree = apply_mutator(json_tree, swap_venue)
```

```
tree_to_string(mutated_tree)
```

```
'{”conference”: ”ICST”}'
```

2 References