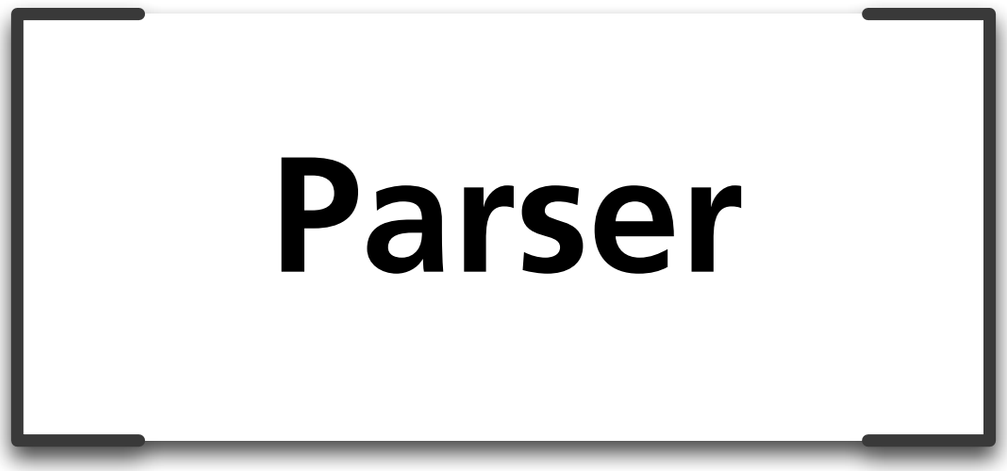
Grammar-Based Fuzzing

Security Testing

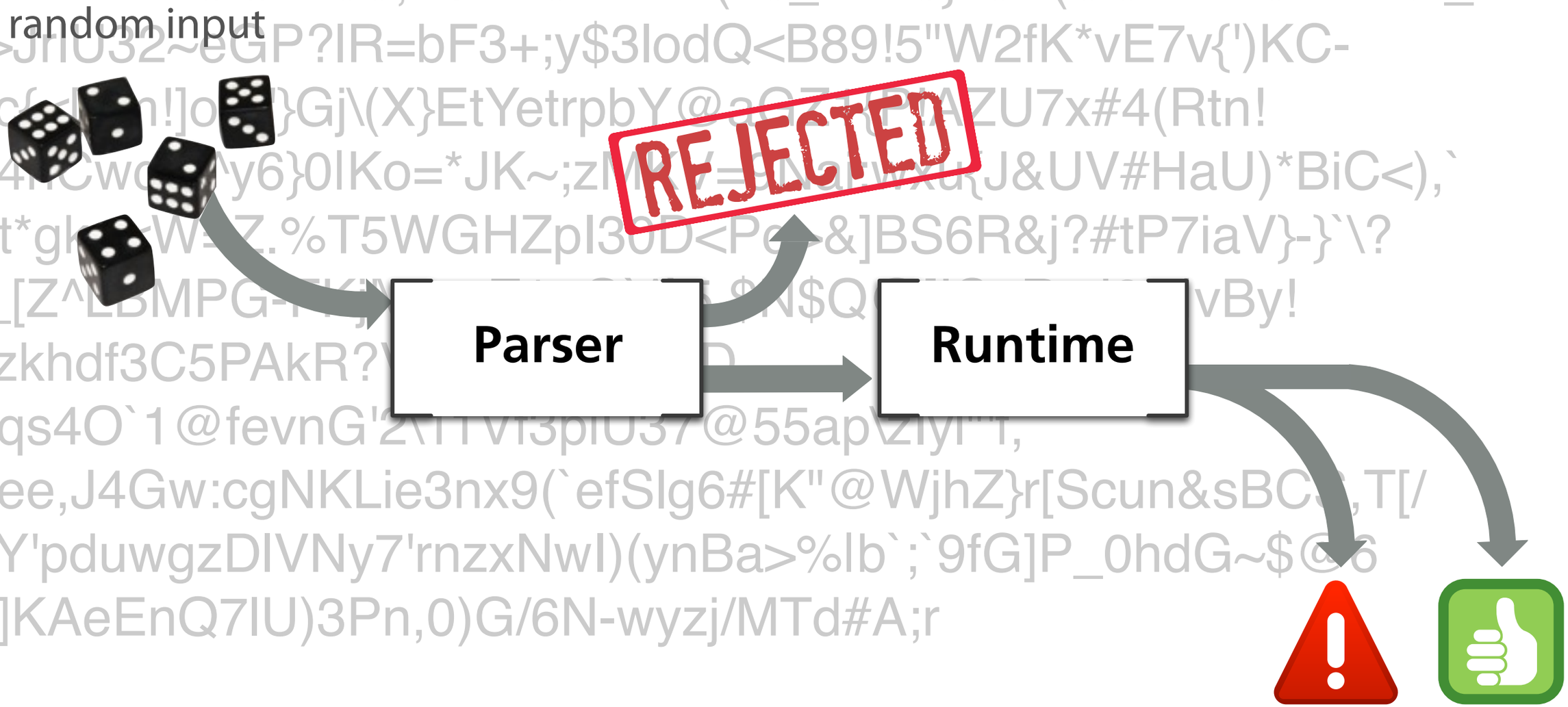
Andreas Zeller, Saarland University

Grammar Fuzzing

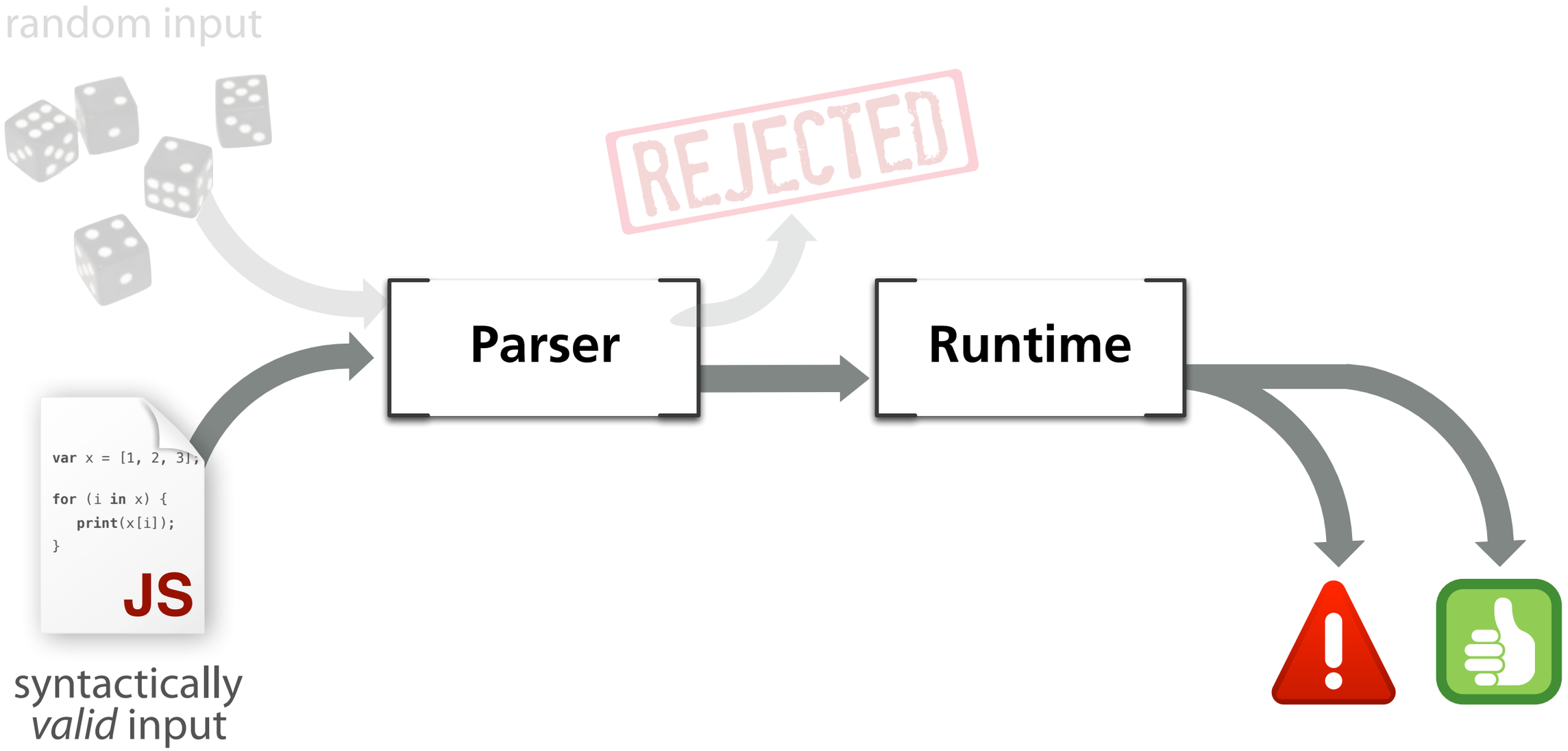
- Suppose you want to test a *parser* – to compile and execute a program



Grammar Fuzzing



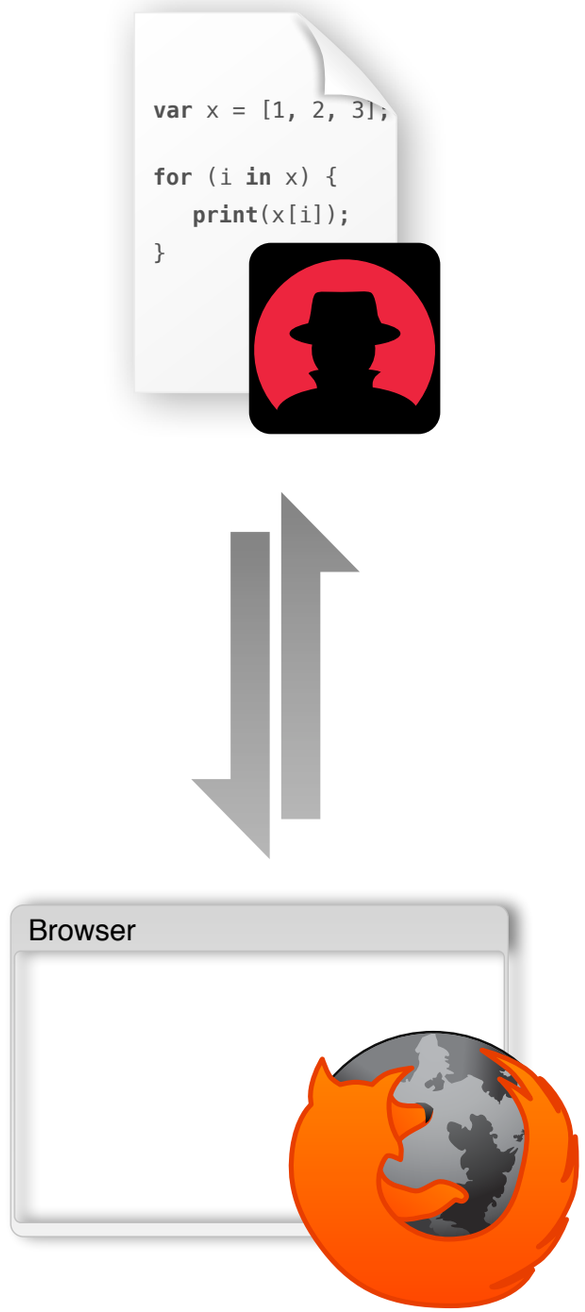
Grammar Fuzzing



LangFuzz

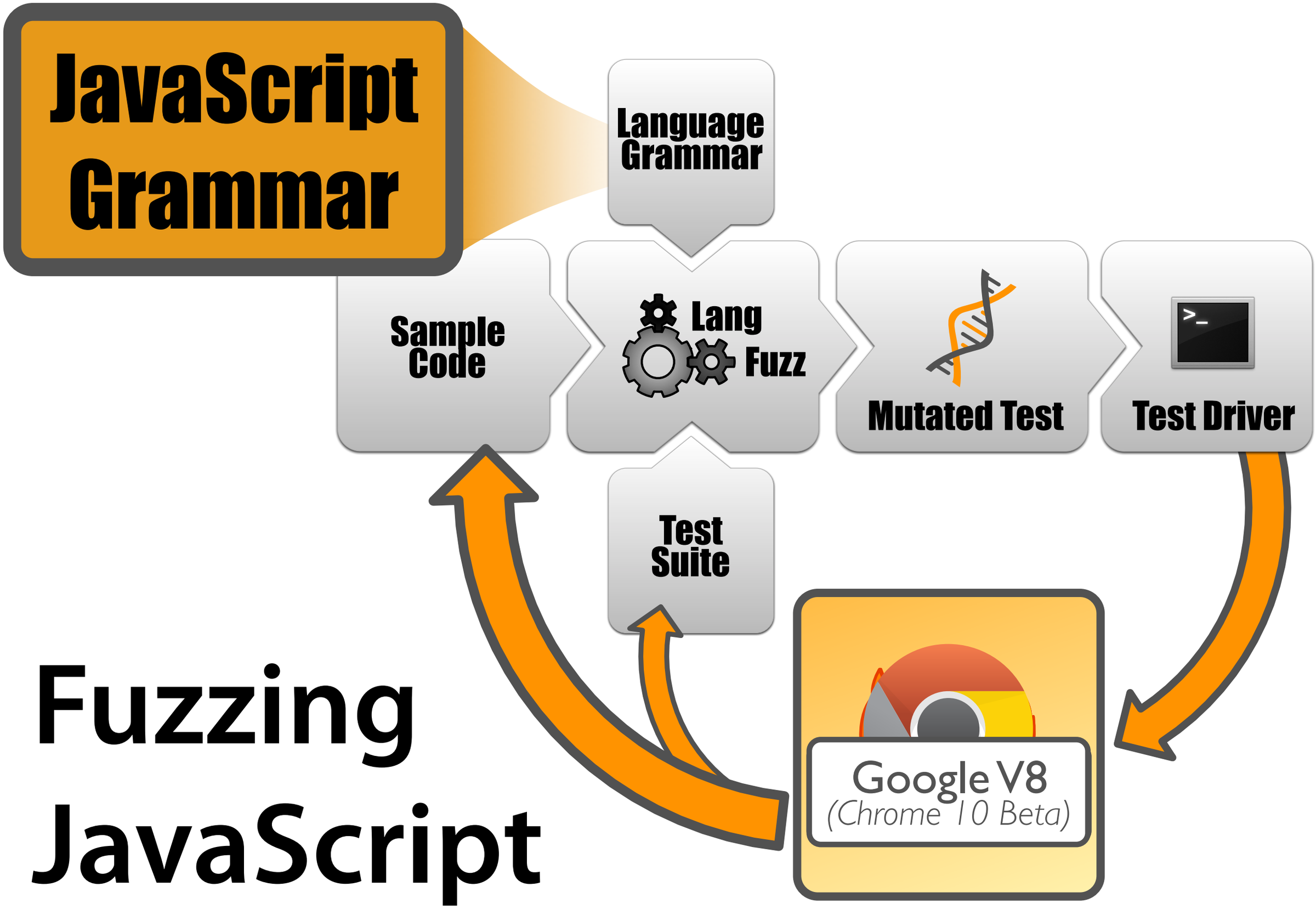
- Fuzz tester using a full-fledged *grammar* to generate inputs
- Can be parametrized with a *grammar*
- Can use grammar to *parse existing inputs*

JavaScript as Domain



- If an attacker gains control over the *JavaScript interpreter*, he gains control over the *entire browser*

Fuzzing JavaScript



JavaScript Grammar

If Statement

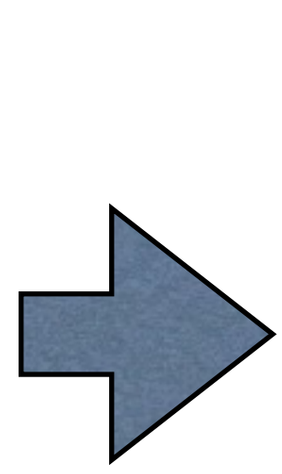
IfStatement[full] ⇒
  `if` *ParenthesizedExpression* *Statement*[full]
| **`if`** *ParenthesizedExpression* *Statement*[noShortIf] **`else`** *Statement*[full]
IfStatement[noShortIf] ⇒ **`if`** *ParenthesizedExpression* *Statement*[noShortIf] **`else`** *Statement*[noShortIf]

Switch Statement

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

A Generated Input

```
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 with Grammars

- Want to encode a *grammar* to produce arithmetic expressions as *strings*
- $START expands into $EXPR, which can expand into $TERM, $TERM + $TERM, etc.

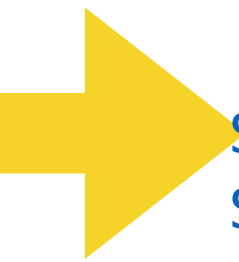
Fuzzing with Grammars

- Want to encode a *grammar* to produce arithmetic expressions as *strings*
- $START expands into $EXPR, which can expand into $TERM, $EXPR + $TERM, etc.

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

$START



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

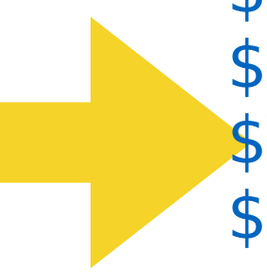
Fuzzing with Grammars

$EXPR

```
$START  ::= $EXPR
$EXPR   ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM   ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR ::= +$FACTOR | -$FACTOR | ($EXPR) |
            $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

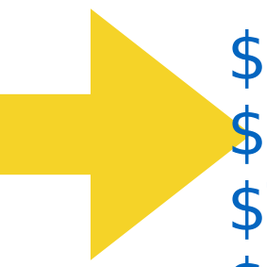
```
$EXPR + $TERM
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

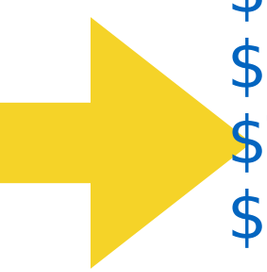
```
$EXPR + $FACTOR
```



```
$START    ::= $EXPR
$EXPR     ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM     ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR   ::= +$FACTOR | -$FACTOR | ($EXPR) |
              $INTEGER | $INTEGER.$INTEGER
$INTEGER  ::= $INTEGER$DIGIT | $DIGIT
$DIGIT    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

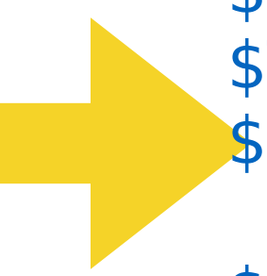
```
$TERM + $FACTOR
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

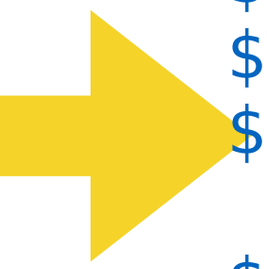
```
$FACTOR + $FACTOR
```



```
$START    ::= $EXPR
$EXPR     ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM     ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR   ::= +$FACTOR | -$FACTOR | ($EXPR) |
              $INTEGER | $INTEGER.$INTEGER
$INTEGER  ::= $INTEGER$DIGIT | $DIGIT
$DIGIT    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

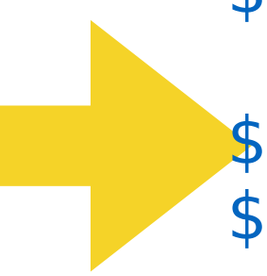
```
$FACTOR + $INTEGER
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

```
$INTEGER + $INTEGER
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
             $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

```
$DIGIT + $INTEGER
```

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

```
2 + $INTEGER
```

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
             $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Fuzzing with Grammars

2 + 2 ✅

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Automatic Production

- Implement production in Python
- Start with $START, apply rules randomly

Grammar Encoding

```
#!/usr/bin/env python
# Grammar-based Fuzzing

import random

term_grammar = {
    "$START":
        ["$EXPR"],

    "$EXPR":
        ["$EXPR + $TERM", "$EXPR - $TERM", "$TERM"],

    "$TERM":
        ["$TERM * $FACTOR", "$TERM / $FACTOR", "$FACTOR"],

    "$FACTOR":
        ["+$FACTOR", "-$FACTOR", "($EXPR)", "$INTEGER",
"$INTEGER.$INTEGER"],

    "$INTEGER":
        ["$INTEGER$DIGIT", "$DIGIT"],

    "$DIGIT":
        ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
```

Demo

grammar-fuzz.py

- Want to encode a *grammar* to produce arithmetic expressions as *strings*
  - $START expands into $EXPR, which can expand into $TERM, $TERM + $TERM, etc.

```
#!/usr/bin/env python
# Grammar-based Fuzzing

import random

term_grammar = {
    "$START":
        ["$EXPR"],
```

```
#!/usr/bin/env python
# Grammar-based Fuzzing

import random

term_grammar = {
    "$START":
        ["$EXPR"],

    "$EXPR":
        ["$EXPR + $TERM", "$EXPR - $TERM", "$TERM"],

    "$TERM":
        ["$TERM * $FACTOR", "$TERM / $FACTOR", "$FACTOR"],

    "$FACTOR":
        ["+$FACTOR", "-$FACTOR", "($EXPR)", "$INTEGER",
"$INTEGER.$INTEGER"],

    "$INTEGER":
        ["$INTEGER$DIGIT", "$DIGIT"],

    "$DIGIT":
        ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
```

```
    "$EXPR":
        ["$EXPR + $TERM", "$EXPR - $TERM", "$TERM"],

    "$TERM":
        ["$TERM * $FACTOR", "$TERM / $FACTOR", "$FACTOR"],

    "$FACTOR":
        ["+$FACTOR", "-$FACTOR", "($EXPR)", "$INTEGER",
"$INTEGER.$INTEGER"],

    "$INTEGER":
        ["$INTEGER$DIGIT", "$DIGIT"],

    "$DIGIT":
        ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}

def apply_rule(term, rule):
    (old, new) = rule
    # We replace the first occurrence;
    # this could also be some random occurrence
    return term.replace(old, new, 1)

MAX_SYMBOLS = 5
MAX_TRIES = 500
```

```
def produce(grammar):
    term = "$START"
    tries = 0

    while term.count('$') > 0:
        # All rules have the same chance;
        # this could also be weighted
        key = random.choice(grammar.keys())
        repl = random.choice(grammar[key])
        new_term = apply_rule(term, (key, repl))
        if new_term != term and new_term.count('$') <
MAX_SYMBOLS:
            term = new_term
            tries = 0
        else:
            tries += 1
            if tries >= MAX_TRIES:
                assert False, "Cannot expand " + term

    return term

if __name__ == "__main__":
    print(produce(html_grammar))
```

$EXPR
$EXPR – $TERM
$EXPR + $TERM – $TERM
$EXPR + $TERM * $FACTOR – $TERM
$TERM + $TERM * $FACTOR – $TERM
$TERM + $TERM * –$FACTOR – $TERM
$FACTOR + $TERM * –$FACTOR – $TERM
–$FACTOR + $TERM * –$FACTOR – $TERM
—$FACTOR + $TERM * –$FACTOR – $TERM
—$FACTOR + $FACTOR * –$FACTOR – $TERM
——$FACTOR + $FACTOR * –$FACTOR – $TERM
——$FACTOR + $FACTOR * –$FACTOR – $FACTOR
——+$FACTOR + $FACTOR * –$FACTOR – $FACTOR
——+–$FA
——+–$IN
——+–$DI
——+–2 +
——+–2 + $INTEGER.$INTEGER * –$FACTOR – $FACTOR
——+–2 + $INTEGER.$INTEGER * –+$FACTOR – $FACTOR
——+–2 + $INTEGER.$INTEGER * –+$INTEGER – $FACTOR
——+–2 + $DIGIT.$INTEGER * –+$INTEGER – $FACTOR
——+–2 + 3.$INTEGER * –+$INTEGER – $FACTOR
——+–2 + 3.$INTEGER * –+$INTEGER – +$FACTOR
——+–2 + 3.$INTEGER * –+$INTEGER – +$INTEGER
——+–2 + 3.$DIGIT * –+$INTEGER – +$INTEGER
——+–2 + 3.5 * –+$INTEGER – +$INTEGER
——+–2 + 3.5 * –+$DIGIT – +$INTEGER
——+–2 + 3.5 * –+1 – +$INTEGER
——+–2 + 3.5 * –+1 – +$DIGIT
——+–2 + 3.5 * –+1 – +5

——+–2 + 3.5 * –+1 – +5

Grammar Coverage

- Idea: Want to *cover* as many rules as possible
- Approach: Save that expansion has been applied (*covered*)
- Prefer *uncovered* over *covered* expansions

Grammar Coverage

$START

```
$START    ::= $EXPR
$EXPR     ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM     ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR   ::= +$FACTOR | -$FACTOR | ($EXPR) |
                  $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

$EXPR

```
$START  ::= $EXPR ✅
$EXPR   ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM   ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

$EXPR + $TERM

```
$START   ::= $EXPR ✅
$EXPR    ::= $EXPR + $TERM ✅ | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

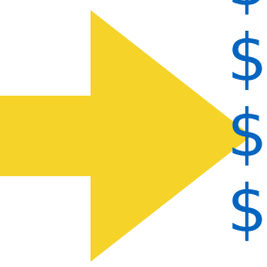
Grammar Coverage

$EXPR + $FACTOR

```
$START   ::= $EXPR ✅
$EXPR    ::= $EXPR + $TERM ✅ | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR ✅
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
             $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

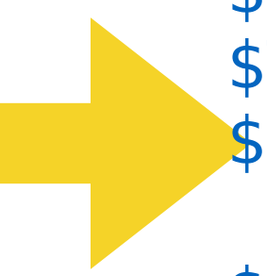
```
$TERM + $FACTOR
```



```
$START   ::= $EXPR ✅
$EXPR    ::= $EXPR + $TERM ✅ | $EXPR - $TERM | $TERM ✅
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR ✅
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

```
$TERM * $FACTOR + $FACTOR
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

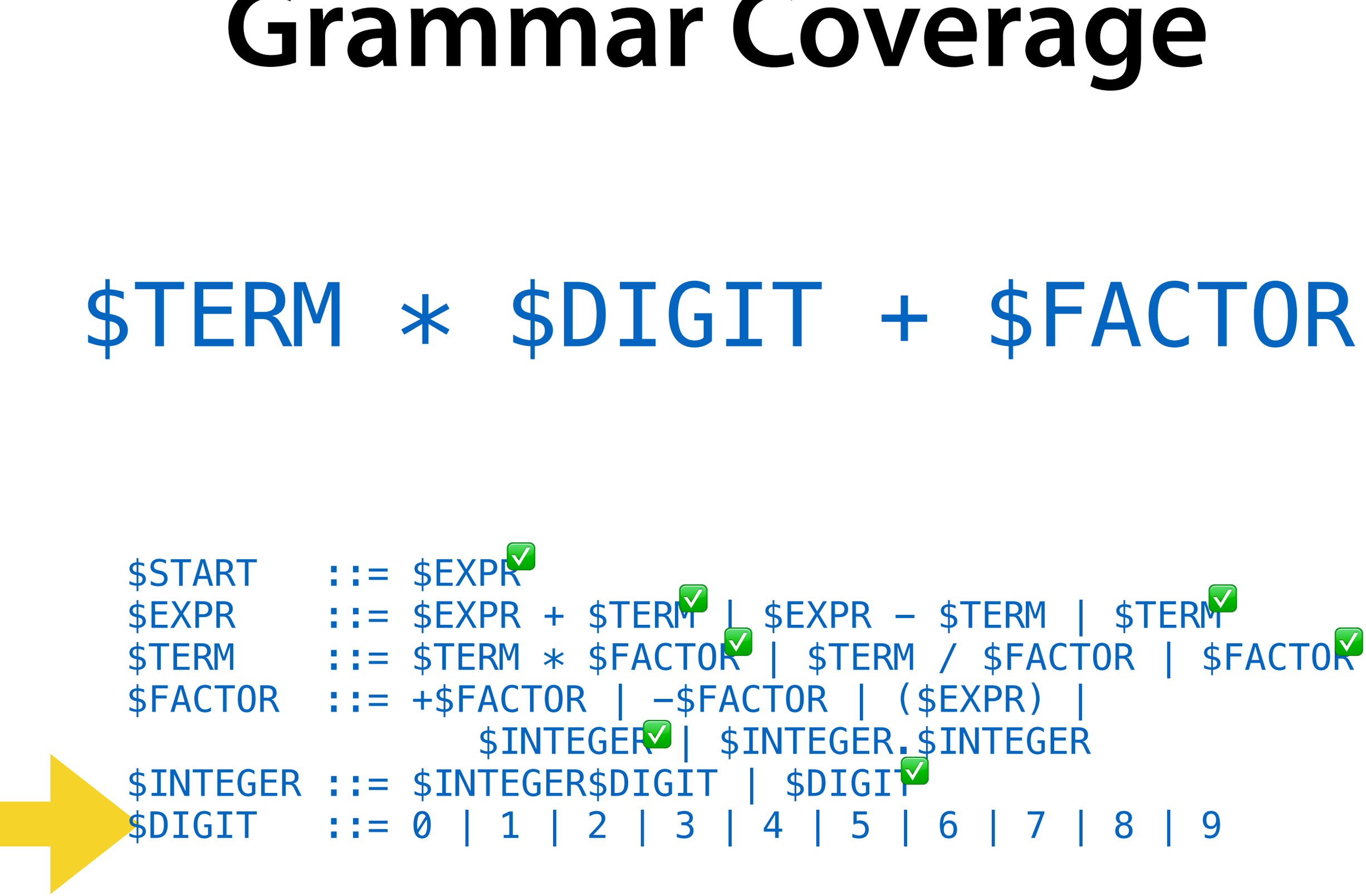
Grammar Coverage

$TERM * $INTEGER + $FACTOR

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

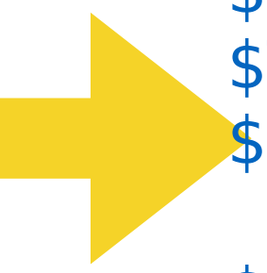
Grammar Coverage

$TERM * $DIGIT + $FACTOR



Grammar Coverage

```
$TERM * 2 + $FACTOR
```



```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

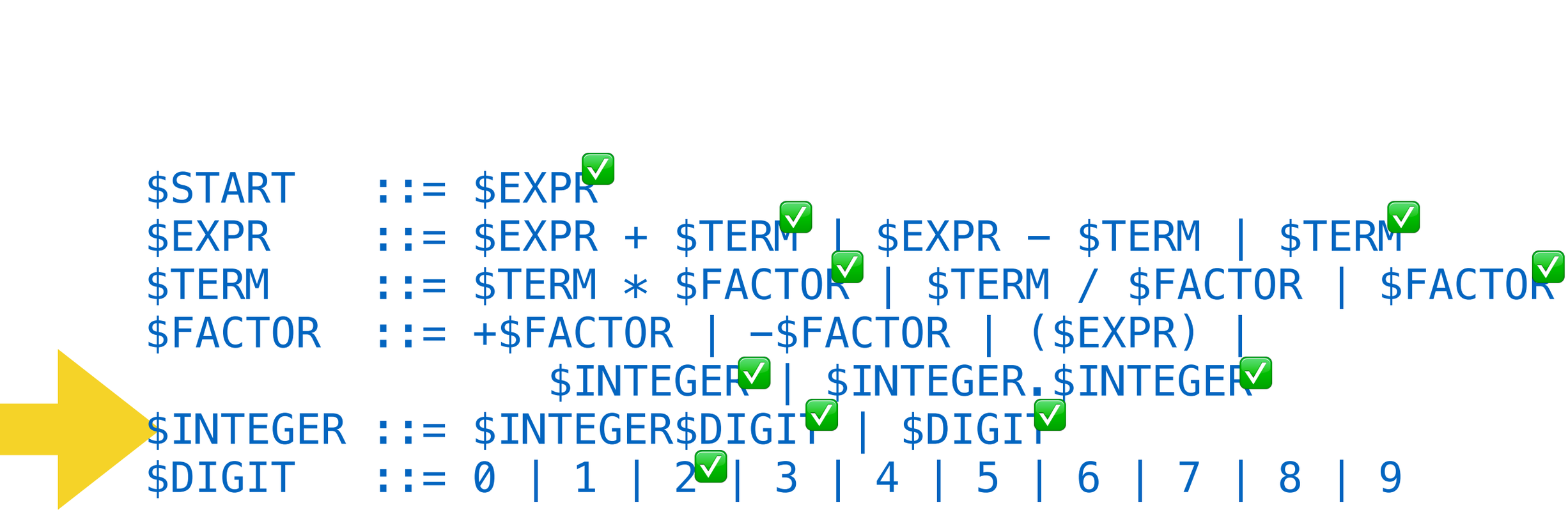
Grammar Coverage

```
$TERM * 2 + $INTEGER.$INTEGER
```

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

```
$TERM * 2 + $INTEGER$DIGIT.$INTEGER
```



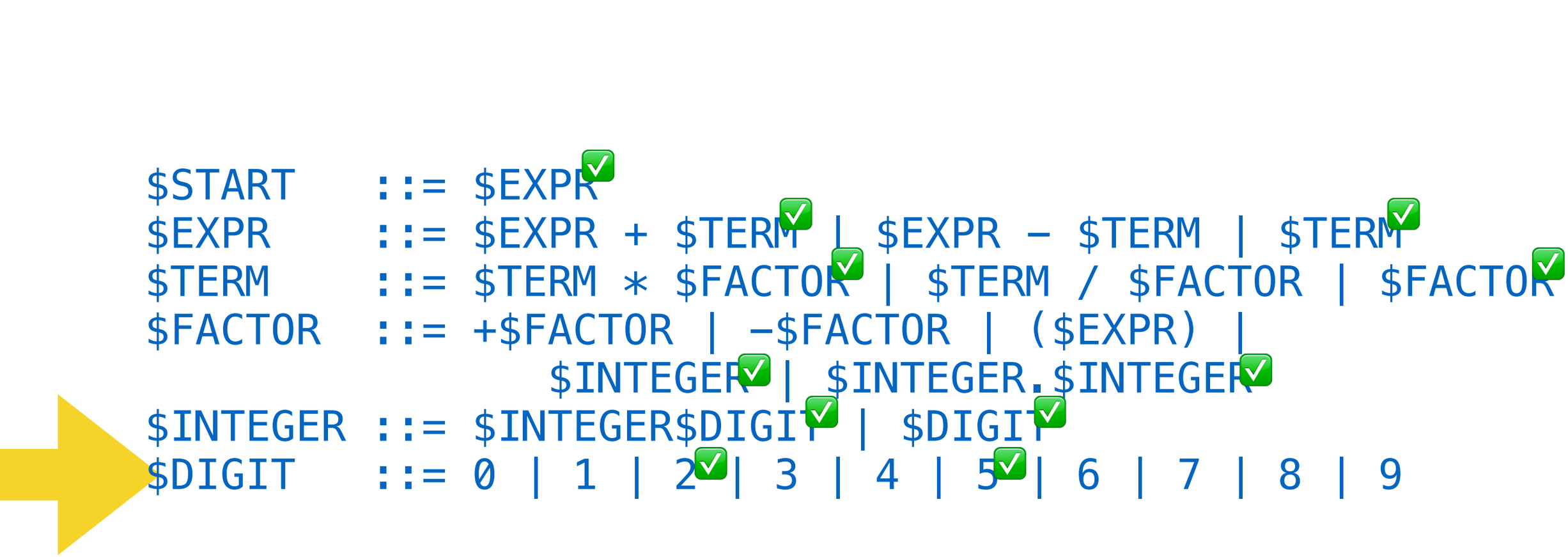
Grammar Coverage

```
$TERM * 2 + $DIGIT$DIGIT.$INTEGER
```

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Grammar Coverage

```
$TERM * 2 + 5$DIGIT.$INTEGER
```



Grammar Coverage

```
$TERM * 2 + 56.$INTEGER
```

```
$START   ::= $EXPR ✅
$EXPR    ::= $EXPR + $TERM ✅ | $EXPR - $TERM | $TERM ✅
$TERM    ::= $TERM * $FACTOR ✅ | $TERM / $FACTOR | $FACTOR ✅
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) |
                $INTEGER ✅ | $INTEGER.$INTEGER ✅
$INTEGER ::= $INTEGER$DIGIT ✅ | $DIGIT ✅
$DIGIT   ::= 0 | 1 | 2 ✅ | 3 | 4 | 5 ✅ | 6 ✅ | 7 | 8 | 9
```

Grammar Coverage

```
−8 / +7 * 2 + 56.9
```

```
$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR − $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | −$FACTOR | ($EXPR) |
                $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Tracking Coverage

- Track coverage in Python
- Mark used productions as "covered"
- Prefer uncovered productions over covered ones

Demo

Things To Do

- Don't apply rules randomly;
  instead, find those that can be applied
- Prefer uncovered *rules* over covered ones
- Prefer rules that *lead* to uncovered ones
- Cover not only single rules, but *pairs*
  (triples? paths?) of expansions
- Produce *multiple outputs*,
  preserving coverage