CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Advances in Language-Based Testing

Andreas Zeller • 50. Treffen der Fachgruppe TAV der GI • 2024-11-14

The two Shades of Testing

Tester

Program

The two Shades of Testing

Tester

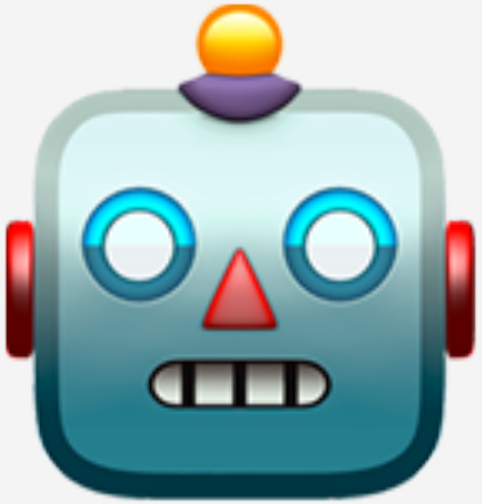
Test against *implementation*

Program

Program

Test against *specification*

The two Shades of Testing



Tester

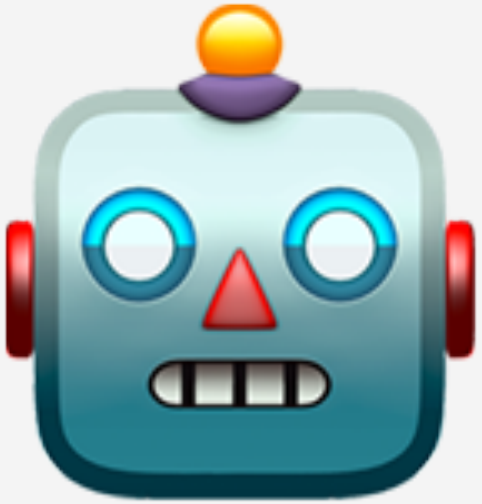
Test against *implementation*

Program

Program

Test against *specification*

Fuzzing



Fuzzer

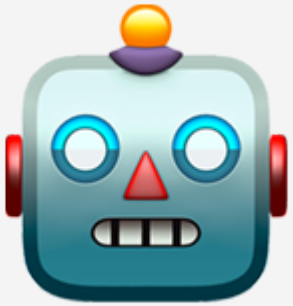
Test against *implementation*

Program

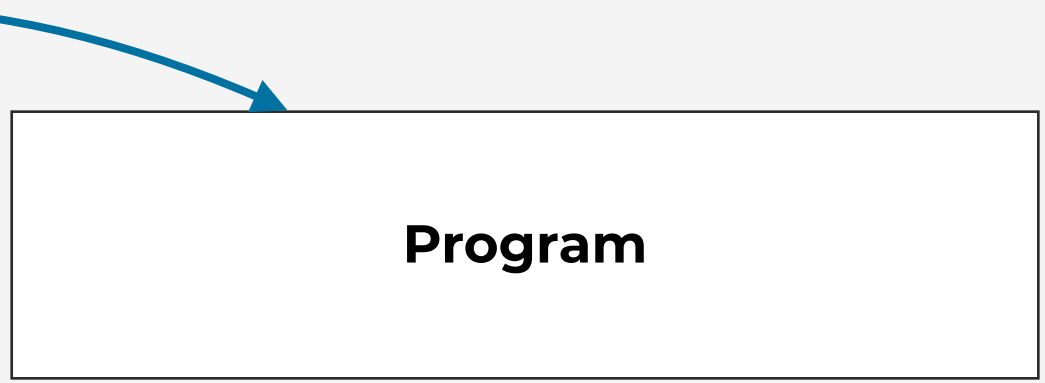
Fuzzing

```
>>> import random; ''.join([chr(random.randint(32, 179))
...     for i in range(100)])
```

```
n}]`@|cq\' clqXc/ ^&M3!w^k>1Qa2%&"`&y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?2\x7fg,+|
U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!s\'m#4nsvDrz>\x7fK| _BA?
lsfZ=(VcS]u;X>x5Kz/<BfpGge#C z8z-pbeNOkgxDlMfl^mT}?$"57bqT Y#VdI\\Y_/$QNQY%8Buzxd354s6T\
\T&XuLOWjhld4S]/-NuwXk+ln3%_RJ|S_q6::{OyjVJP4(%bTy(.mA@-`:=ms5~w(3!o/_D#1\\h4N>^b|
vCXWeKZ|v(qj`Ci.C+]1&vU*Ed[*"H"\x7f|\x7f.+i"sD/zLBElwu1~vW~X|Vof?[JlNf8#ouU 6F%D<:^gP-e]-
N.>b0vkU*C+0Ch&Ud0m}Vx2Ibl.*Jp"C}b}<ByN^/|<Hdt"e{}({`_hpB4x+6+HZDJ .Mh3NB;
hgjF3wKGxx1Jj~zRR>
```

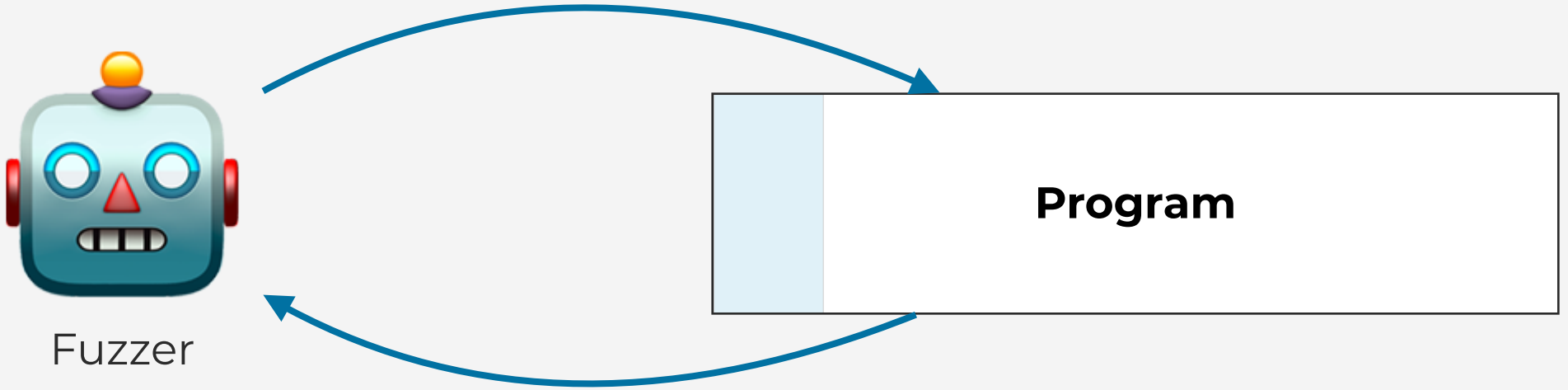


Fuzzer



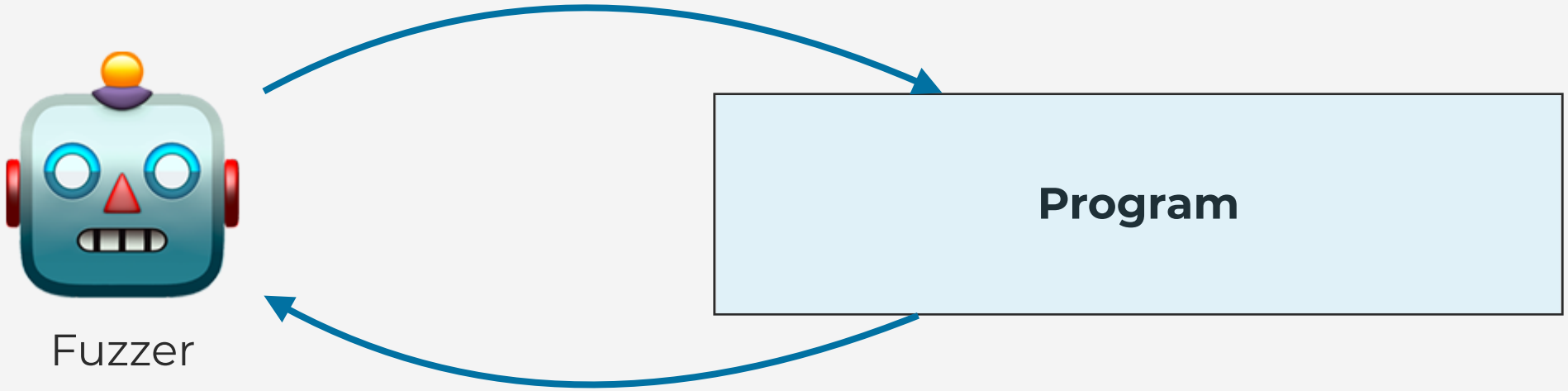
Mutating

```
n}]` $ cq\' clqXc/ ^&M3!w^k>1Qa2%& $ y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?2\x7fg,+|
U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!s\'m#4nsvDrz>\x7fK| _BA?
lsfZ=(VcS]u;X>x5Kz/< < pGge#C z8z-pbeNOkgxD ! fl^mT < $"57bqT Y#VdI\\Y_/$QN ! %8Buzxd354s6T\
\T&XuLOWjhld4S]/-NuwXk+ln3%_ A |S_q6::{OyjVJP4(%bTy(.mA@-` A ns5~w(3!o/_D#1\\h4N A o|
vCXWeKZ|v(qj`Ci.C+]1&vU*Ed[*"H"\x7f|\x7f.+i"sD/zLBElwu1~vW~X|Vof?[JlNf8#ouU 6F%D<:^gP-e]-
N.>b 1 kU*C+0Ch&Ud0m}Vx Z ol.*Jp"C} 1 ByN^/|<Hdt"e{}( Z hpB4x+6+HZDJ .Mh3N Z
hgjF3wKGxx1Jj~zRR>
```



Guidance

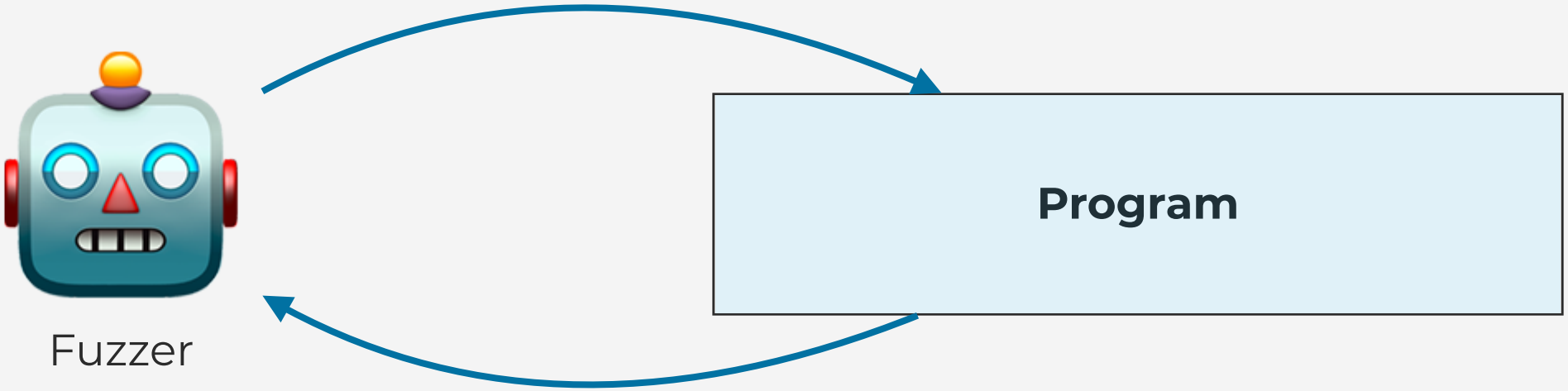
```
n}]` $ cq\' clqXc/ ^&M3!w^k>1Qa2%& $ y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?2\x7fg,+|
U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!s\'m#4nsvDrz>\x7fK| _BA?
lsfZ=(VcS]u;X>x5Kz/< < pGge#C z8z-pbeNOkgxD ! fl^mT < $"57bqT Y#VdI\\Y_/$QN ! %8Buzxd354s6T\
\T&XuLOWjhld4S]/-NuwXk+ln3%_ A |S_q6::{OyjVJP4(%bTy(.mA@-` A ns5~w(3!o/_D#1\\h4N A o|
vCXWeKZ|v(qj`Ci.C+]1&vU*Ed[*"H"\x7f|\x7f.+i"sD/zLBElwu1~vW~X|Vof?[JlNf8#ouU 6F%D<:^gP-e]-
N.>b 1 kU*C+0Ch&Ud0m}Vx Z ol.*Jp"C} 1 <ByN^/|<Hdt"e{}( Z hpB4x+6+HZDJ .Mh3N Z
hgjF3wKGxx1Jj~zRR>
```



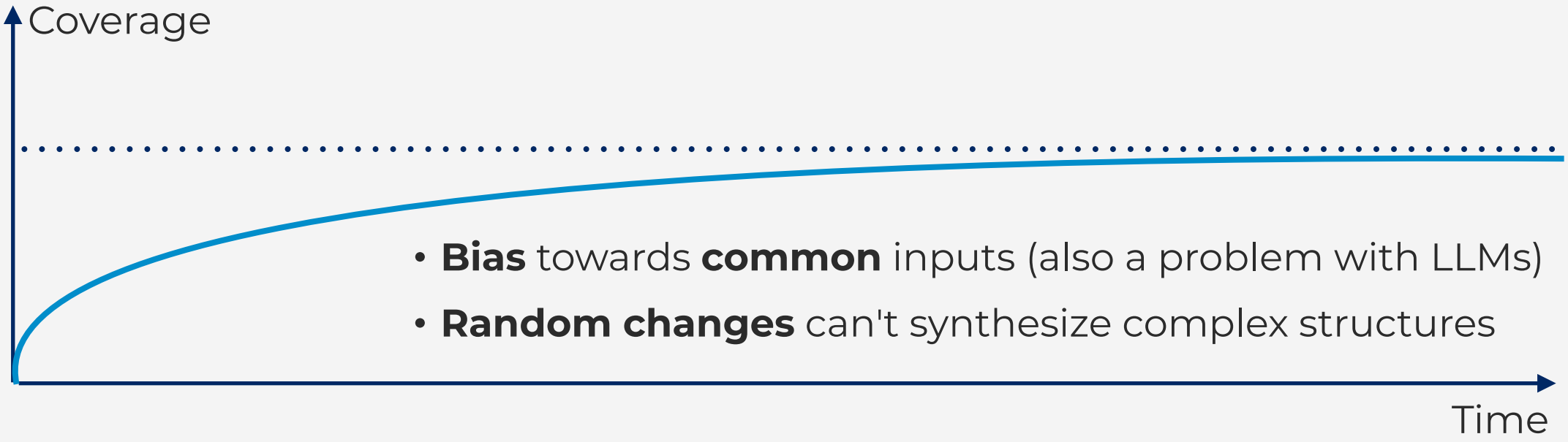
Fuzzing: The #1 Technique to Detect Vulnerabilities

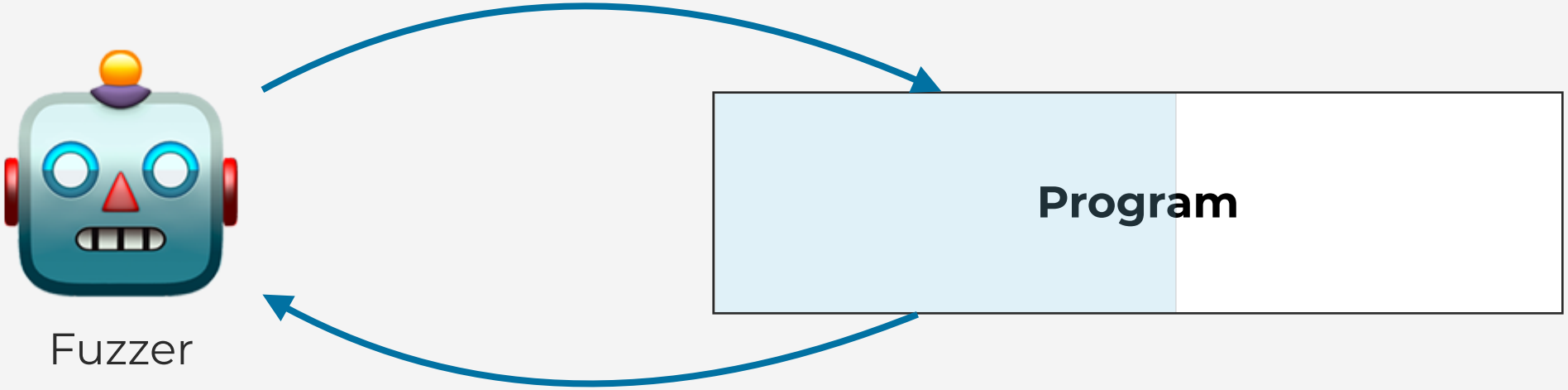
- **Fuzzers** are the most effective technique to find bugs
- **Routinely used** in all big tech companies
- Mostly **"fire-and-forget"**: One fuzzer rules them all

The Coverage Glass Ceiling

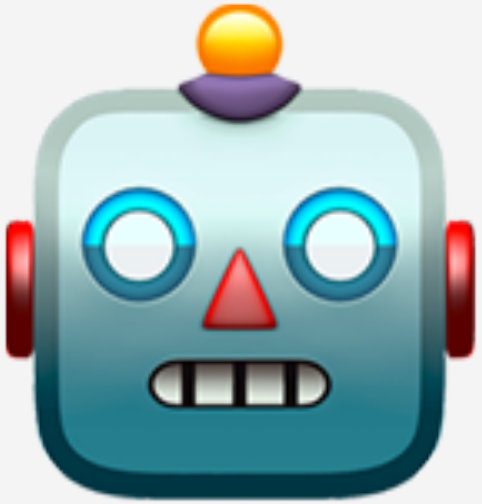


The Coverage Glass Ceiling





The two Shades of Testing



Fuzzer

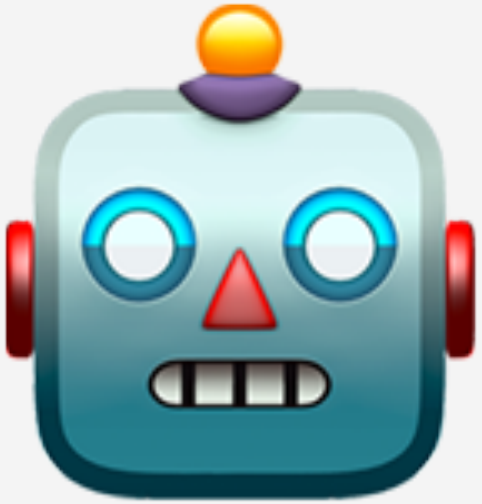
Test against *implementation*

Program

Program

Test against *specification*

The two Shades of Testing



Fuzzer

Program

Test against *specification*

Specifying Inputs: Grammars

Specify a language (= a set of inputs)

Expansion rule

Nonterminal symbol

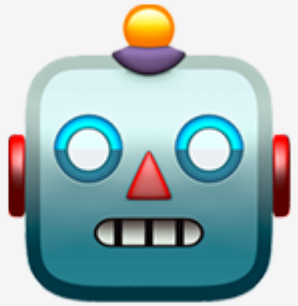
```
⟨start⟩    ::= ⟨expr⟩
⟨expr⟩     ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩     ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩   ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩      ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩    ::= 0 | 1 | 2 | 3 | 4 | 5 |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

Terminal symbol

Producer

Grammars as Producers

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

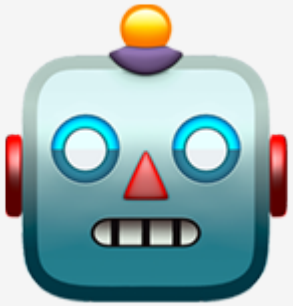


Producer

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



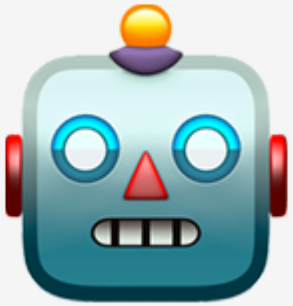
Producer

⟨start⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



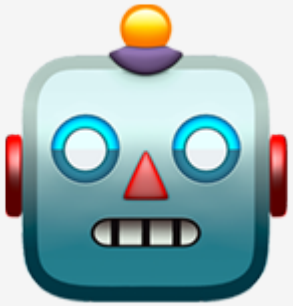
Producer

⟨start⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



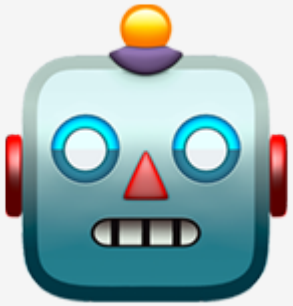
Producer

⟨expr⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



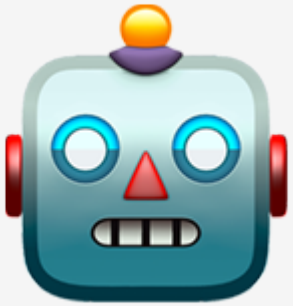
Producer

⟨term⟩ - ⟨expr⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



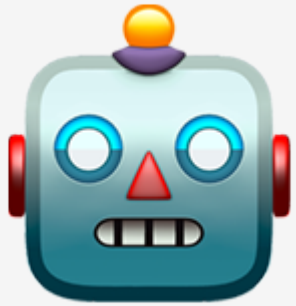
Producer

⟨term⟩ - ⟨expr⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



Producer

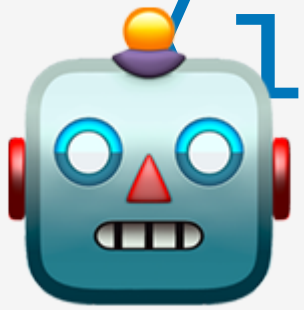
⟨factor⟩ - ⟨expr⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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

⟨int⟩ . ⟨int⟩ - ⟨expr⟩



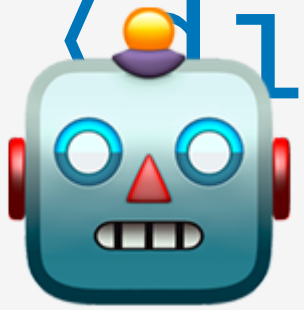
Producer

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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

⟨digit⟩ . ⟨int⟩ - ⟨expr⟩



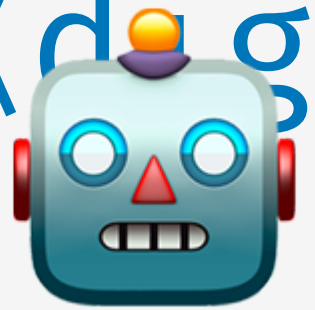
Producer

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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

⟨digit⟩ . ⟨digit⟩ - ⟨expr⟩



Producer

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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

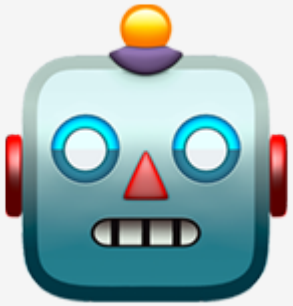
8. ⟨digit⟩ - ⟨expr⟩

Producer

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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



Producer

8.2 – ⟨expr⟩

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

Grammars as Producers

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

```
8.2 - 79 - -9 / +((+9 * --2 + --+-+-((-1 *
```

- **Grammars** are common and well-understood specification method
- Turning a grammar into a **producer** is a simple task
- **LangFuzz** (2012) found 2,600+ JavaScript bugs this way

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.

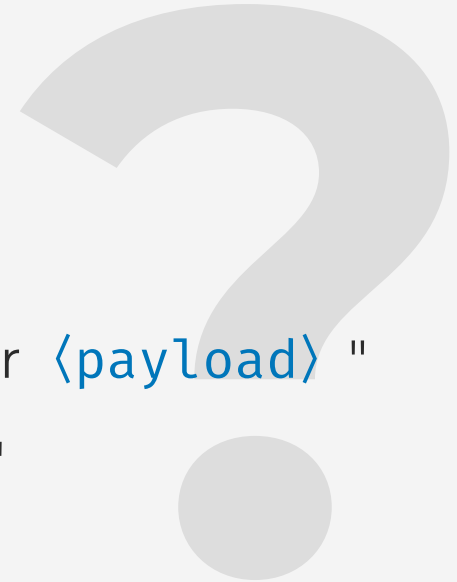
Grammars do not suffice

Syntax
Grammar

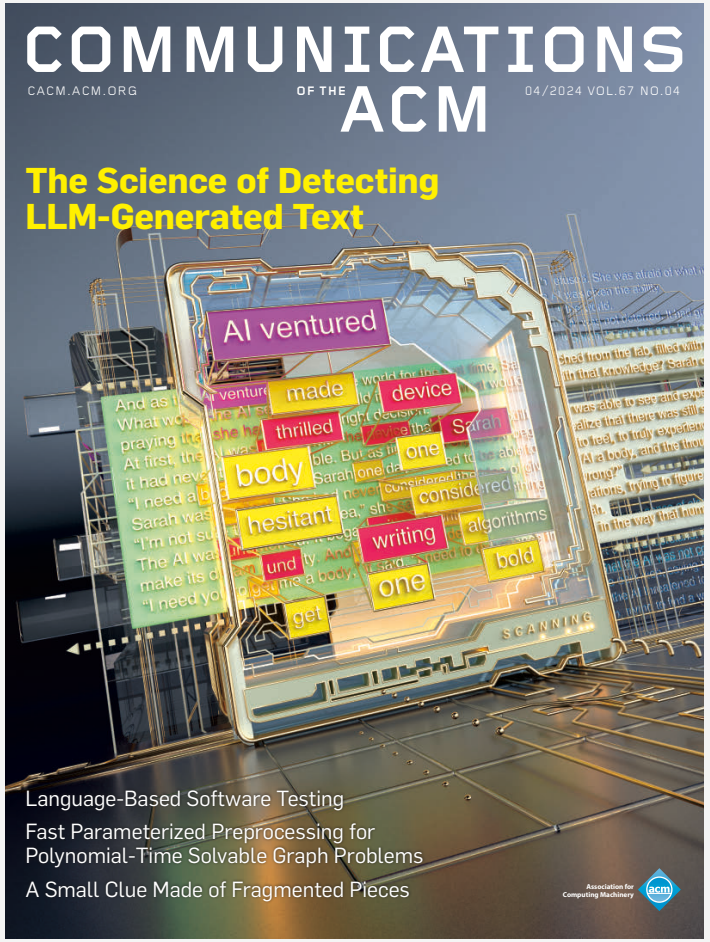
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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

How do you express **semantic properties** such as

- "The *number* should be between 50 and 100"
- "The *length* should be at least 1024 characters"
- (For binary inputs) "The ⟨checksum⟩ should be the *checksum* over ⟨payload⟩ "
- (For code) "Any ⟨identifier⟩ must be *declared* before it is *used*"



In the News



DOI:10.1145/3631520

Constraints over grammar elements can make test generation easier than ever.

BY DOMINIC STEINHÖFEL AND ANDREAS ZELLER

Language-Based Software Testing

SOFTWARE TESTING CONTINUES to be the number one technique to satisfy safety, security, and privacy outputs, relate these to the inputs, and thus explore and check its functionality. This task faces two long-standing challenges.

First, there is the *input generation problem*: How can a bot create inputs that reliably cover functionality? Random system input generators ("fuzzers"[20]) easily detect issues and vulnerabilities in input processing of arbitrary programs. However, creating valid inputs and interactions that reliably reach code beyond input processing is still a challenge for which test generators require human assistance—either through input specifications[1,5,6,10,12,23] or a comprehensive population of diverse input samples that cover behavior.[3,17,19,27]

One might assume that large language models (LLMs) might alleviate this problem. Couldn't a bot learn how to interact with software from examples or documentation? Or what makes a valid input? The problem is that to find bugs, one needs valid but also uncommon inputs—namely, those that trigger less common (and less tested) behavior. LLMs may help find common (and thus valid) inputs—but by construction, they will not find uncommon inputs.

The second challenge is the long-standing *test oracle problem*: How can a bot check the outputs of a system? All test generators and fuzzers assume some oracle that checks for correctness—by default, a generic, implicit oracle detecting illegal or un-

Specifying Languages

Syntax

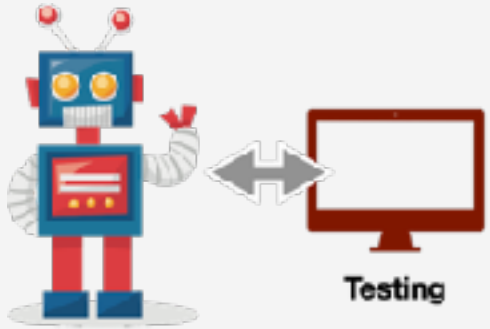
Grammar

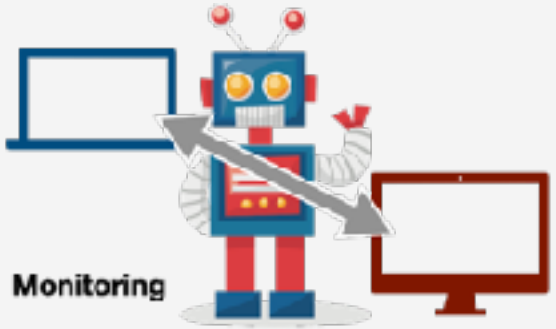
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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

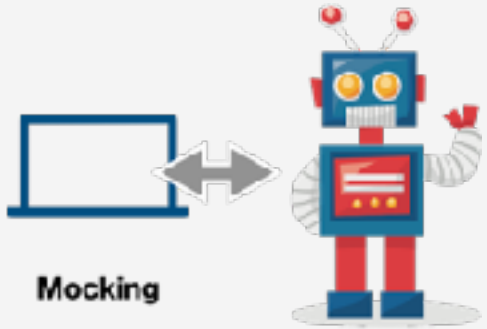
Semantics

Constraints

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```







Specifying Languages

Syntax

Grammar

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

Semantics

Constraints

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

- Solves *constraints symbolically* to produce correct inputs
- Constraints are SMT-LIB functions over nonterminals
- Powerful, but slow and inflexible

Specifying Languages

Syntax

Grammar

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

Semantics

Constraints

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

Fandango

Specifying Languages with Fandango

Syntax

Grammar

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

Semantics

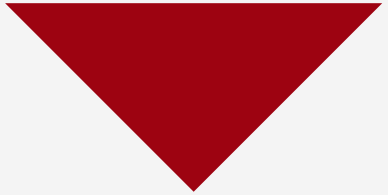
Constraints

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

- Fandango uses **evolutionary testing** to quickly **solve constraints**
- Produces **hundreds of valid inputs** per second
- Unbounded **expressiveness** of constraints

How Fandango works – Initial population

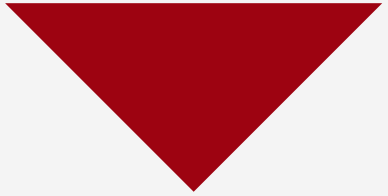
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



```
79 + 7 * -3   811 * (-23 + 4)    -43679.33     8.3
2 + 2     -345 + 080     34578 - (03 - 34798 / 79)
33 / 47.9      5    9023 / (3 + (+4 * 23))     -20
```

How Fandango works – Elite selection

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

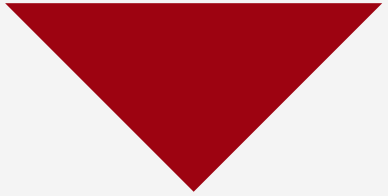


```
79 + 7 * -3   811 * (-23 + 4)    -43679.33     8.3
2 + 2    -345 + 080     34578 - (03 - 34798 / 79)
33 / 47.9      5    9023 / (3 + (+4 * 23))     -20
```

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

How Fandango works – Crossover

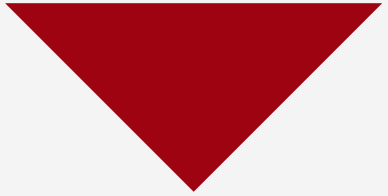
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



```
79 + 7 * -3  811 * (-23 + 4)
                    34578 - (03 - 34798 / 79)
                 9023 / (3 + (+4 * 23))
```

How Fandango works – Crossover

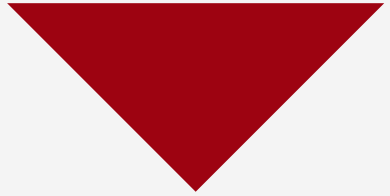
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



```
79 + 7 * -3   811 * (-23 + 4)   (-23 + 4) * (34798 / 79)
                      34578 - (03 - 34798 / 79)
                   9023 / (3 + (+4 * 23))
```

How Fandango works – Mutation

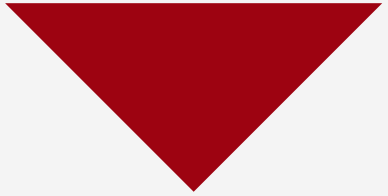
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



```
79 + 7 * -3  811 * (-23 + 4)  (-23 + 4) * (34798 / 79)
                    34578 - (03 - 34798 / 79)
                  9023 / (3 + (+4 * 23))
```

How Fandango works – Mutation

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



```
79 + 7 * -3   811 * (-23 + 4)   (-23 + 4) * (34798 / 79)
                      34578 - (03 - 34798 / 79)
                (-79 - 529) / (3 + (+4 * 23))
```

How Fandango works – Evolution

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

Elite Selection

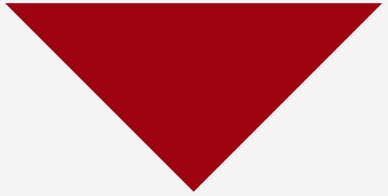
Crossover

Mutation

```
 79 + 7 * -3   811 * (-23 + 4)   (-23 + 4) * (34798 / 79)
34578 - (03 - -56)       34578 - (03 - 34798 / 79)
34578 - 811 * -3    (-79 - 529) / (3 + (+4 * 23))
```

How Fandango works – Solution

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



```
79 + 7 * 3
```

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

Evaluation of Fandango

Subject	Tool	#Inputs per second	Precision	Coverage 4-path	Speedup	Mean length	Median length
CSV 1.0	FANDANGO	**1791.56**	100.00%	100.00%	**866×**	10.49	12.00
(PYTHON csv)	ISLa	2.06	100.00%	100.00%		1212.33	837.00
reST 0.21.2	FANDANGO	**814.58**	100.00%	**100.00%**	**100×**	12.18	10.00
(PYTHON docutils)	ISLa	8.18	100.00%	95.13%		31.96	32.00
ScriptSizeC 0.9.28rc	FANDANGO	**141.75**	100.00%	**100.00%**	**21×**	23.75	24.00
(PYTHON tccbox)	ISLa	6.85	100.00%	55.67%		29.05	30.00
TAR 3.5.3	FANDANGO	**13.33**	100.00%	**100.00%**	**48×**	2048.00	2048.00
(bsdtar)	ISLa	0.28	100.00%	87.42%		4025.03	4096.00
XML 1.3.0	FANDANGO	**71.66**	100.00%	**100.00%**	**31×**	21.38	23.00
(PYTHON xml)	ISLa	2.38	100.00%	90.50%		45.10	49.00

An XML statement

```
<?xml version="1.0" encoding="windows-1251"?>
<hash>cbbceebacec2befcbafebcff</hash>
<statement>
   <info>
      <currency>USD</currency>
      <stmt_date>1729104800</stmt_date>
      <amount>5000</amount>
   </info>
   <sender>
      <name>THOMAS DAVIS</name>
      <account_no>883237407835108666431 6</account_no>
      <bank_key>69567</bank_key>
      <start_balance>8000</start_balance>
      <end_balance>3000</end_balance>
   </sender>
   <receiver>
      <name>SARAH BROWN</name>
      <account_no>40183311208778798612 80</account_no>
      <bank_key>12333</bank_key>
      <start_balance>0</start_balance>
      <end_balance>5000</end_balance>
   </receiver>
</statement>
```

An XML attack

```
<?xml version="1.0" encoding="windows-1251"?>
<hash>cbbceebacec2befcbafebcff</hash>
<statement>
   <info>
      <currency>USD</currency>
      <stmt_date>1729104800</stmt_date>
      <amount>5000</amount>
   </info>
   <sender>
      <name>A LONG NAME THAT CAUSES A BUFFER OVERFLOW
      <account_no>883237407835108666431 6</account_no>
      <bank_key>69567</bank_key>
      <start_balance>8000</start_balance>
      <end_balance>3000</end_balance>
   </sender>
   <receiver>
      <name>SARAH BROWN</name>
      <account_no>40183311208778798612 80</account_no>
      <bank_key>12333</bank_key>
      <start_balance>0</start_balance>
      <end_balance>5000</end_balance>
   </receiver>
</statement>
```

An XML attack

```
<?xml version="1.0" encoding="windows-1251"?>
<hash>cbbceebacec2befcbafebcff</hash>
<statement>
   <info>
      <currency>'); DROP TABLE CUSTOMERS; --</currency
      <stmt_date>1729104800</stmt_date>
      <amount>5000</amount>
   </info>
   <sender>
      <name>A LONG NAME THAT CAUSES A BUFFER OVERFLOW
      <account_no>883237407835108666431 6</account_no>
      <bank_key>69567</bank_key>
      <start_balance>8000</start_balance>
      <end_balance>3000</end_balance>
   </sender>
   <receiver>
      <name>SARAH BROWN</name>
      <account_no>40183311208778798612 80</account_no>
      <bank_key>12333</bank_key>
      <start_balance>0</start_balance>
      <end_balance>5000</end_balance>
   </receiver>
</statement>
```

An XML attack

```
<?xml version="1.0" encoding="windows-1251"?>
<hash>cbbceebacec2befcbafebcff</hash>
<statement>
   <info>
      <currency>'); DROP TABLE CUSTOMERS; --</currency
      <stmt_date>1729104800</stmt_date>
      <amount>5000</amount>
   </info>
   <sender>
      <name>A LONG NAME THAT CAUSES A BUFFER OVERFLOW
      <account_no>883237407835108666431 6</account_no>
      <bank_key>69567</bank_key>
      <start_balance>NaN</start_balance>
      <end_balance>NaN</end_balance>
   </sender>
   <receiver>
      <name>SARAH BROWN</name>
      <account_no>40183311208778798612 80</account_no>
      <bank_key>12333</bank_key>
      <start_balance>0</start_balance>
      <end_balance>5000</end_balance>
   </receiver>
</statement>
```

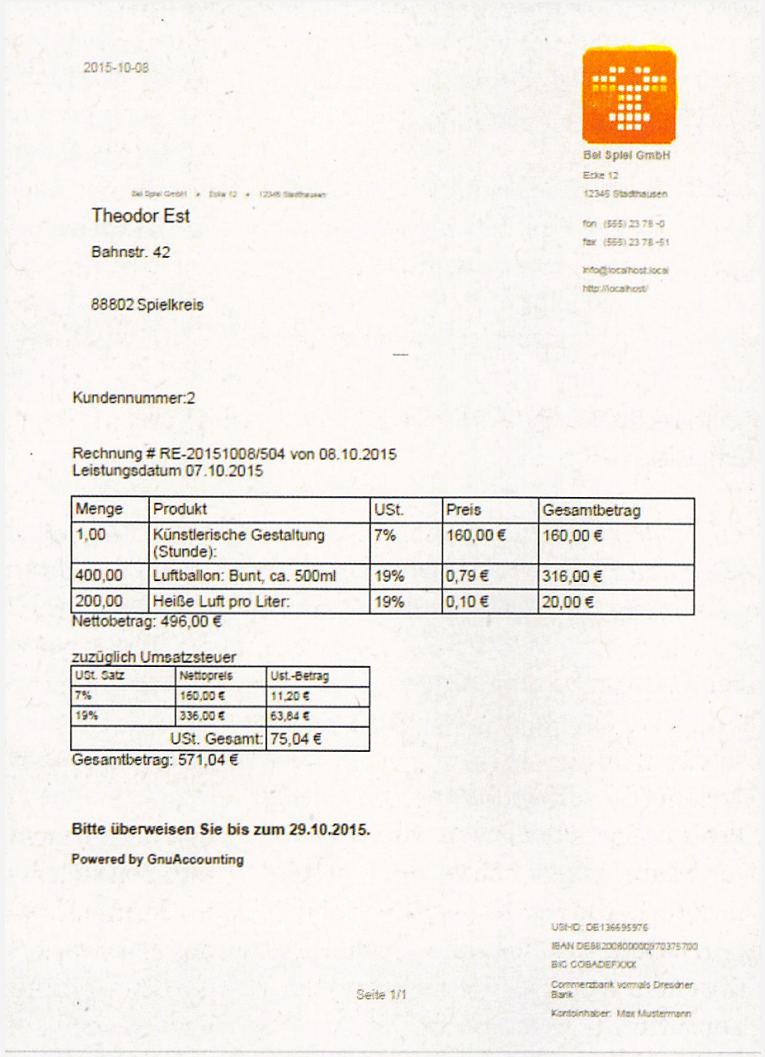
XML Constraints

```
BR-CO-13: (ram:TaxBasisTotalAmount = ram:LineTotalAmount -
ram:AllowanceTotalAmount + ram:ChargeTotalAmount) or
((ram:TaxBasisTotalAmount = ram:LineTotalAmount -
ram:AllowanceTotalAmount) and not (ram:ChargeTotalAmount)) or
((ram:TaxBasisTotalAmount = ram:LineTotalAmount +
ram:ChargeTotalAmount) and not (ram:AllowanceTotalAmount)) or
((ram:TaxBasisTotalAmount = ram:LineTotalAmount) and not
(ram:ChargeTotalAmount) and not (ram:AllowanceTotalAmount))
```

BR-CO-13: Der Inhalt des Elementes „Invoice total amount without VAT“ (BT-109) entspricht der Summe aller Inhalte der Elemente „Invoice line net amount“ (BT-131) abzüglich der Summe aller in der Rechnung enthaltenen Nachlässe der Dokumentenebene „Sum of allowances on document level“ (BT-107) zuzüglich der Summe aller in der Rechnung enthaltenen Abgaben der Dokumentenebene „Sum of charges on document level“ (BT-108).

- XML has **constraints** that protect against invalid data
- But what if you can **solve** these constraints as we do?

Example: XRechnung



2015-10-08

Bei Spiel GmbH
Ecke 12
12345 Stadthausen

fon (555) 23 78 -0
fax (555) 23 78 -51

info@localhost.local
http://localhost/

Theodor Est

Bahnstr. 42

88802 Spielkreis

Kundennummer:2

Rechnung # RE-20151008/504 von 08.10.2015
Leistungsdatum 07.10.2015

Menge	Produkt	USt.	Preis	Gesamtbetrag
1,00	Künstlerische Gestaltung (Stunde):	7%	160,00 €	160,00 €
400,00	Luftballon: Bunt, ca. 500ml	19%	0,79 €	316,00 €
200,00	Heiße Luft pro Liter:	19%	0,10 €	20,00 €

Nettobetrag: 496,00 €

zuzüglich Umsatzsteuer

USt. Satz	Nettopreis	Ust.-Betrag
7%	160,00 €	11,20 €
19%	336,00 €	63,84 €
USt. Gesamt:		75,04 €

Gesamtbetrag: 571,04 €

Bitte überweisen Sie bis zum 29.10.2015.

Powered by GnuAccounting

USt-ID: DE136695976
IBAN DE88200800000970375700
BIC COBADEFXXX
Commerzbank vormals Dresdner Bank
Kontoinhaber: Max Mustermann

Seite 1/1

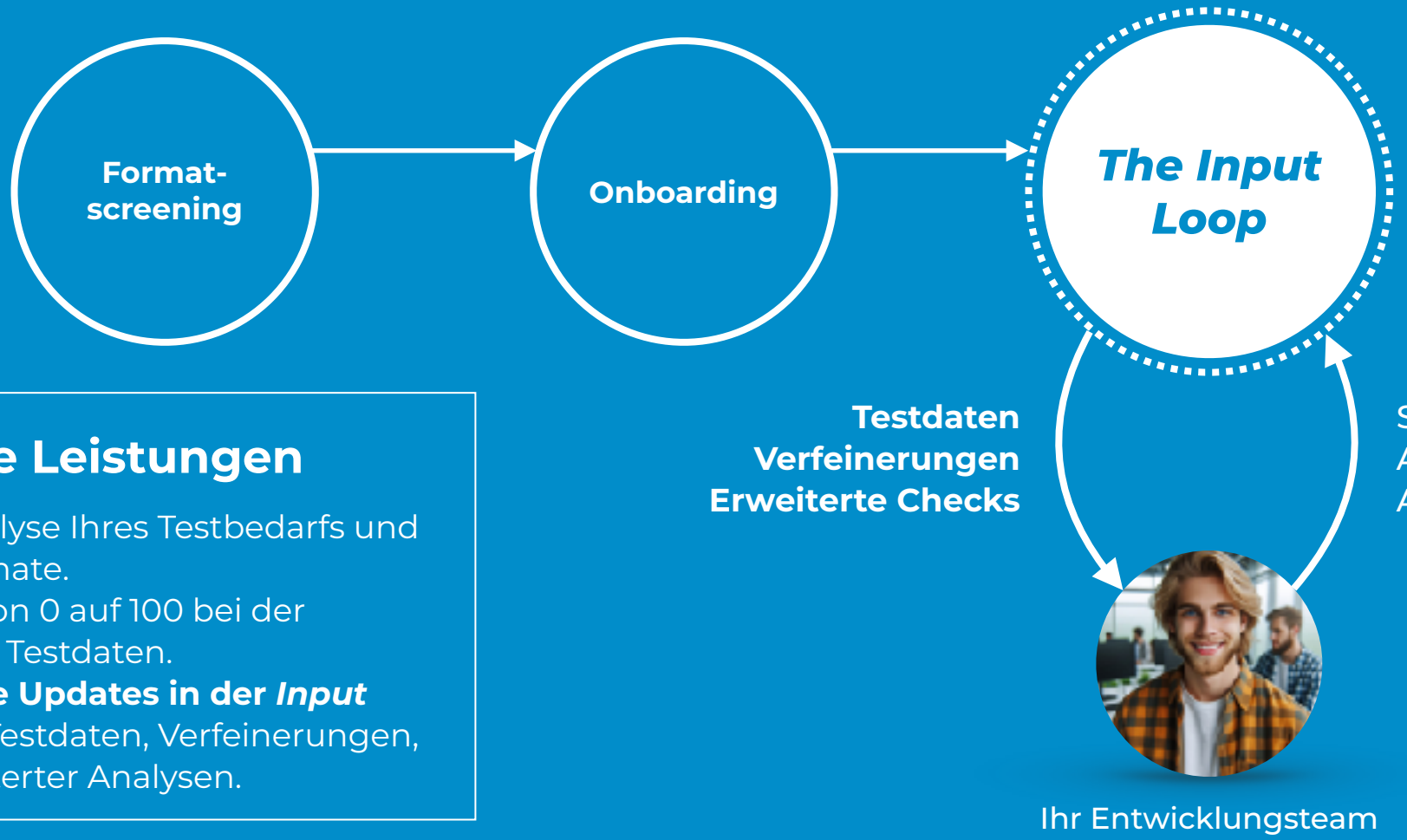
- **Standard** for electronic invoices
- **180** data fields
- Official test suite has **60** handwritten XML invoices
- None of these test for **uncommon values**
- Becomes **mandatory** in 2025
- **What could possibly go wrong?**

Our Solution: Selling Test Data

Our Solution: Selling Test Data

- Creates and sells **synthetic test data** (in bulk and to measure)
- **GDPR**-compliant, privacy-friendly
- Focus on **XML** and other schema-based formats

Die ***InputLoop***. Wie wir zusammen Ihre Systeme verbessern.



Unsere Leistungen

- **Screening**: Analyse Ihres Testbedarfs und Ihrer Datenformate.
- **Onboarding**: Von 0 auf 100 bei der Integration von Testdaten.
- **Kontinuierliche Updates in der *Input Loop***: Weitere Testdaten, Verfeinerungen, Angebot erweiterter Analysen.

inputlab

Roadmap

800k€ for Spin-Off

09/24

Pilot Customers

11/24

800k€ for Start-Up

09/25

Series A

07/26

European Expansion

07/27

inputlab

The two Shades of Testing

Tester

Test against *implementation*

Program

Program

Test against *specification*

Blackbox Testing with Fandango

Syntax

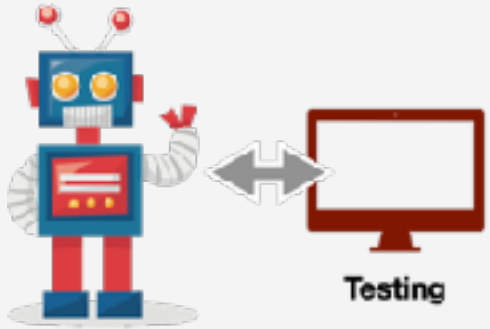
Grammar

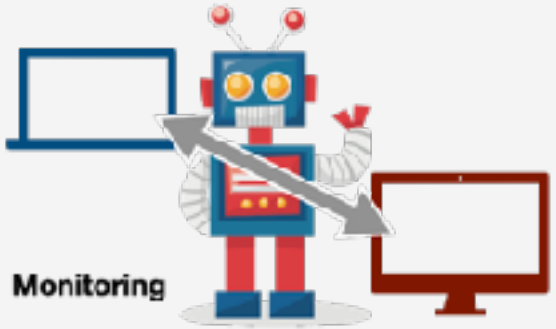
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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

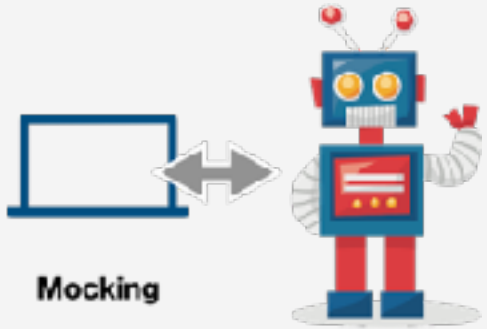
Semantics

Constraints

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```







Whitebox Testing with Fandango

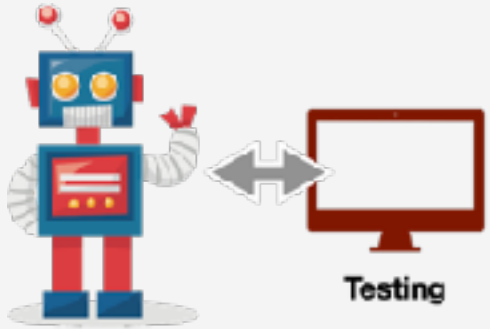
Syntax | **Grammar**

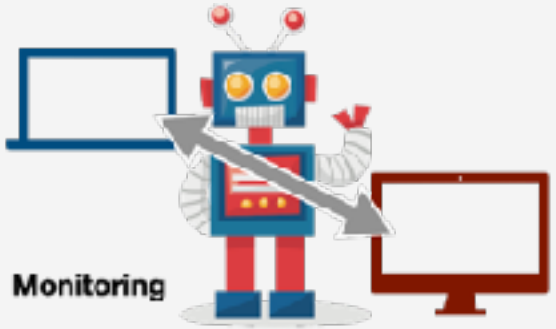
```
⟨start⟩ ::= ⟨byte⟩*
```

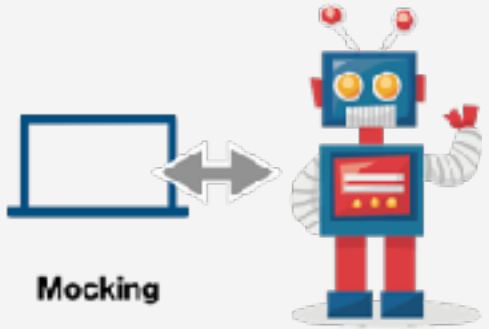
- Mutations at the **byte level**
- **Evolve population** until target reached

Semantics | **Constraints**

```
coverage(⟨start⟩, "myprogram.exe") >= 0.70
```







Whitebox + Blackbox Testing with Fandango

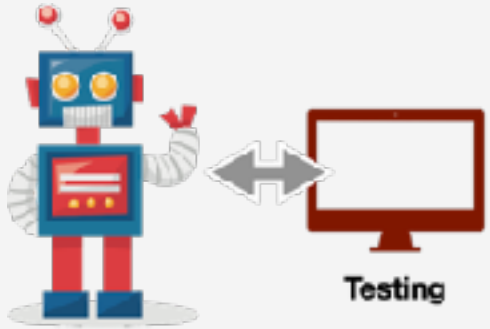
Syntax
Grammar

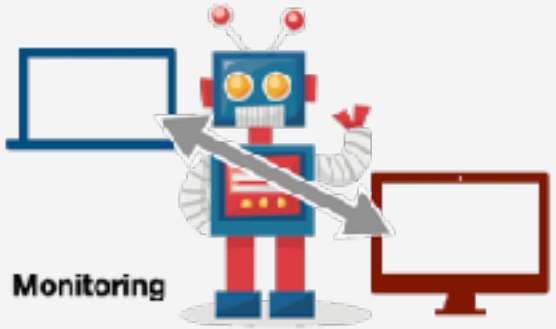
```
⟨start⟩ ::= ⟨chunk⟩*
⟨chunk⟩ ::= ⟨length⟩ ⟨content⟩
```

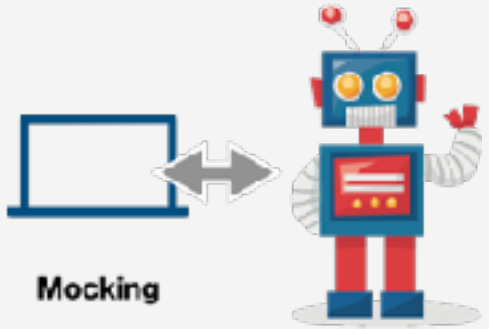
The more you know, the better the test generator

Semantics
Constraints

```
uint16(⟨length⟩) == len(content)
coverage(⟨start⟩, "myprogram.exe") >= 0.70
```







The two Shades of Testing

Tester

Test against *implementation*

Program

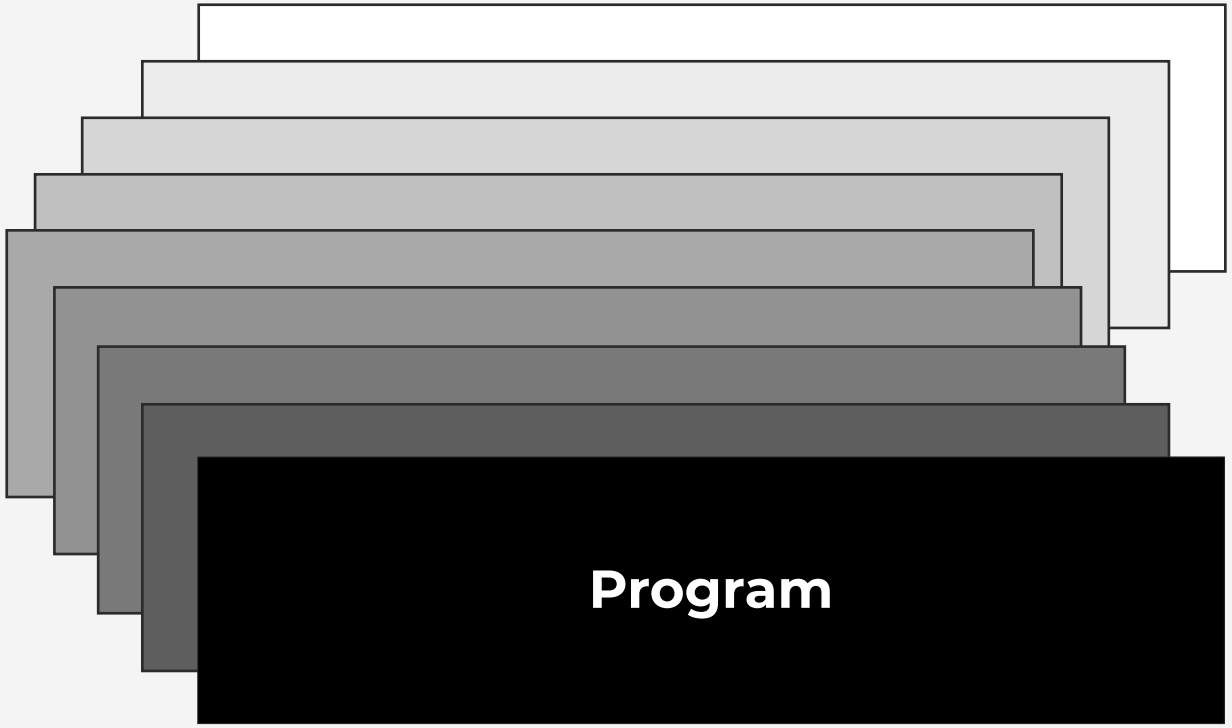
Program

Test against *specification*

Fifty Shades of Greybox Fuzzing

Tester

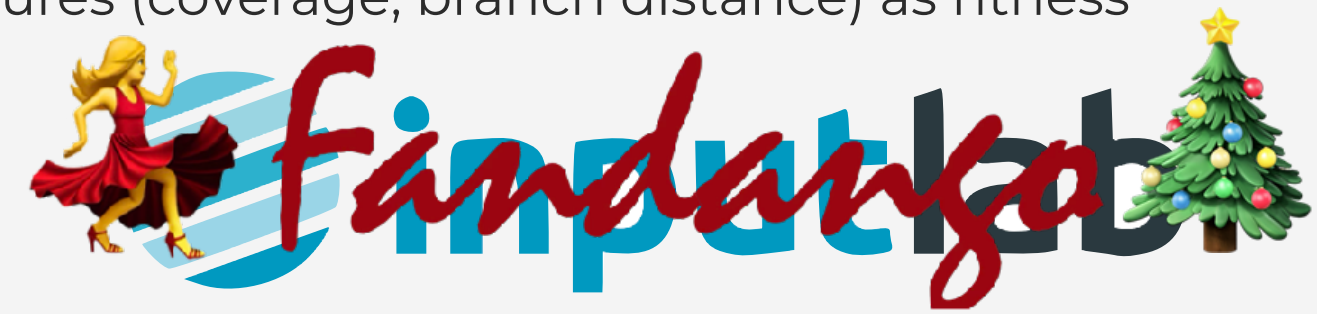
Test against *implementation*



Test against *specification*

Ongoing Fandango works

- **Compilation** – compile a Fandango spec into code, get another 10-100x speedup
- **Monitoring** – use grammar for parsing inputs, check syntax + semantics
- **Binary formats** – specify and test complex formats such as SSL or MP4
- **Protocol testing** – specify and test client/server interactions such as TLS
- **Security testing** – inject malicious payloads all over the place
- **Execution features** – ask for some code to be reached or a loop to be taken
- **Statistical distributions** – ask for test data to be specifically distributed
- **Standard library** – for hashes, encodings, and other frequently needed functions
- **Constraint solving** – integrate symbolic constraint solver for chaotic search spaces
- **AFL-style fuzzing** – use execution features (coverage, branch distance) as fitness
- and many more…



The Fuzzing Book

fuzzingbook.org

The Fuzzing Book | About this Book | Resources | Share | Help

The Fuzzing Book

Tools and Techniques for Generating Software Tests

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

About this Book

Welcome to "The Fuzzing Book"! 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 bookutils import YouTubeVideo
YouTubeVideo('w4u5gCgPlmg')
```

Generating Software Tests

Copy link

Generating Software Tests

Breaking Software for Fun and

Fuzzing: The #1 Technique to Detect Vulnerabilities


- **Fuzzers** are the most effective technique to find bugs
- **Routinely used** in all big tech companies
- Mostly **"fire-and-forget"**: One fuzzer rules them all

x

Grammars as Producers

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

```
8.2 - 79 - -9 / +((+9 * --2 + --+-+-((-1 *
```

- **Grammars** are common and well-understood specification method
- Turning a grammar into a **producer** is a simple task
- **LangFuzz** (2012) found 2,600+ JavaScript bugs this way

Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure**. ASE 2019.

Specifying Languages with Fandango

Syntax (Grammar)

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

Semantics (Constraints)

```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

- Fandango uses **evolutionary testing** to quickly **solve constraints**
- Produces **hundreds of valid inputs** per second
- Unbounded **expressiveness** of constraints

x

Ongoing Fandango works

- **Compilation** – compile a Fandango spec into code, get another 10-100x speedup
- **Monitoring** – use grammar for parsing inputs, check syntax + semantics
- **Binary formats** – specify and test complex formats such as SSL or MP4
- **Protocol testing** – specify and test client/server interactions such as TLS
- **Security testing** – inject malicious payloads all over the place
- **Execution features** – ask for some code to be reached or a loop to be taken
- **Statistical distributions** – ask for test data to be specifically distributed
- **Standard library** – for hashes, encodings, and other frequently needed functions
- **Constraint solving** – integrate symbolic constraint solver for chaotic search spaces
- **AFL-style fuzzing** – use execution features (coverage, branch distance) as fitness

andreas-zeller.info

Fandango