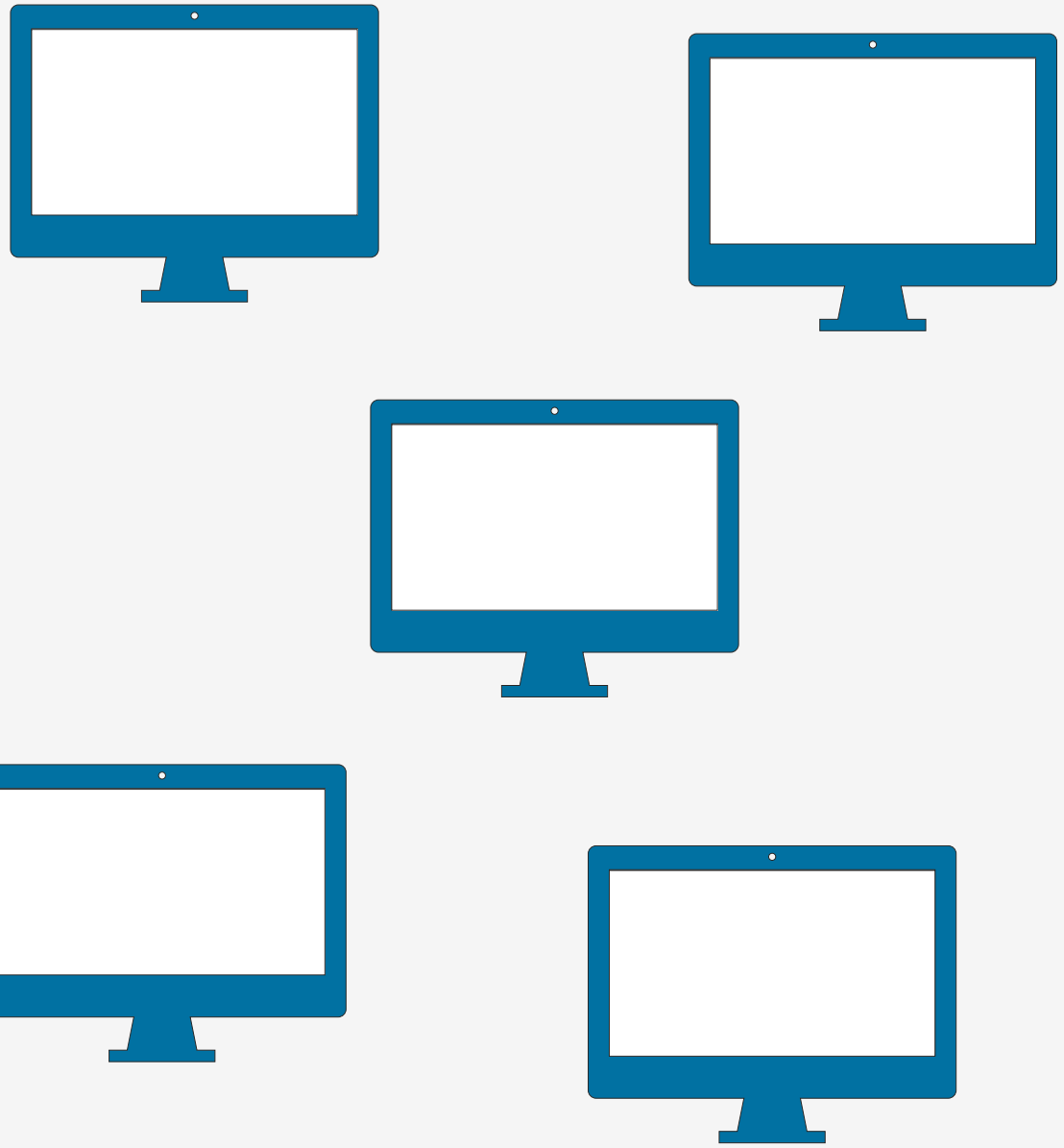
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Generating Test Inputs and Interactions with the  Fandango Fuzzer

Andreas Zeller · GOTO Conference · 2026-09-30

The AI Threat

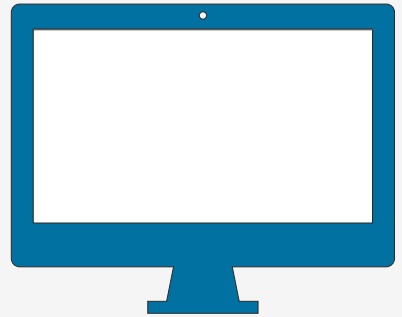


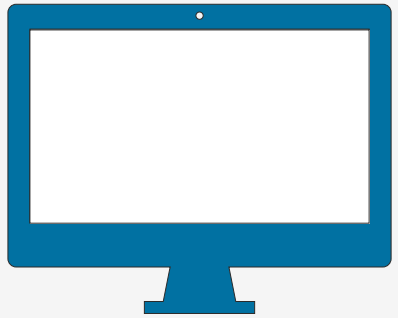


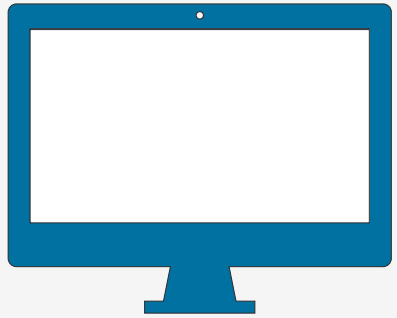
The AI Threat









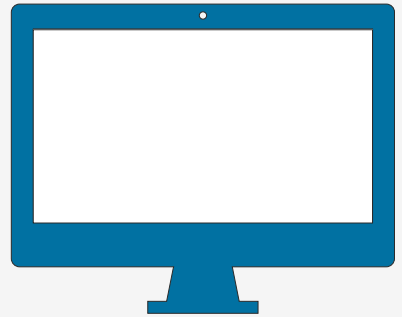


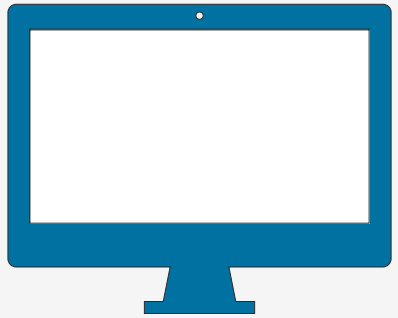


The AI Threat









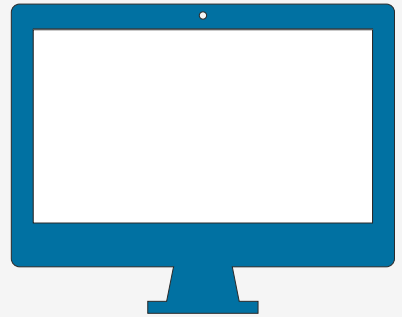




The AI Threat













The AI Threat





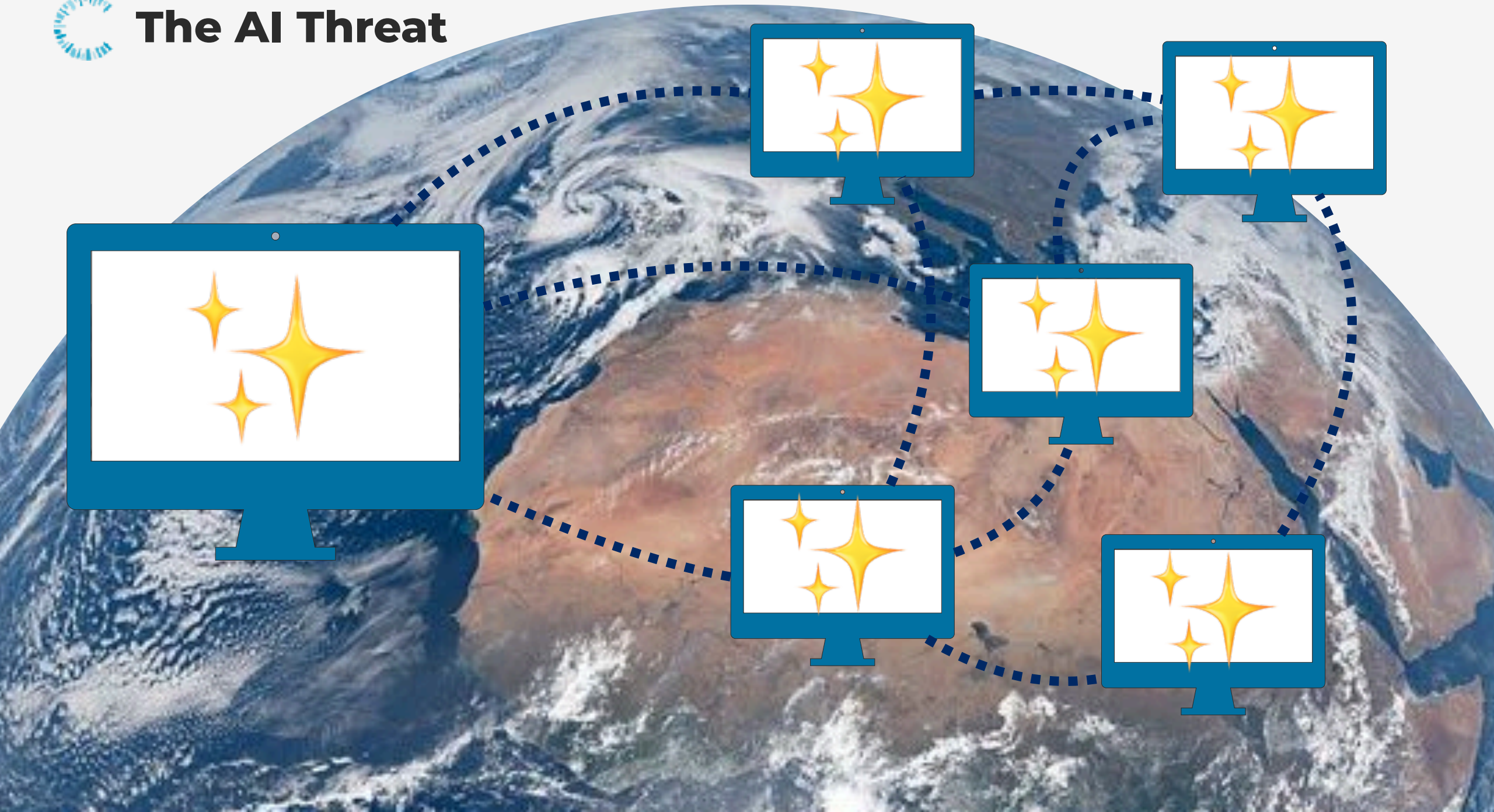








The AI Threat





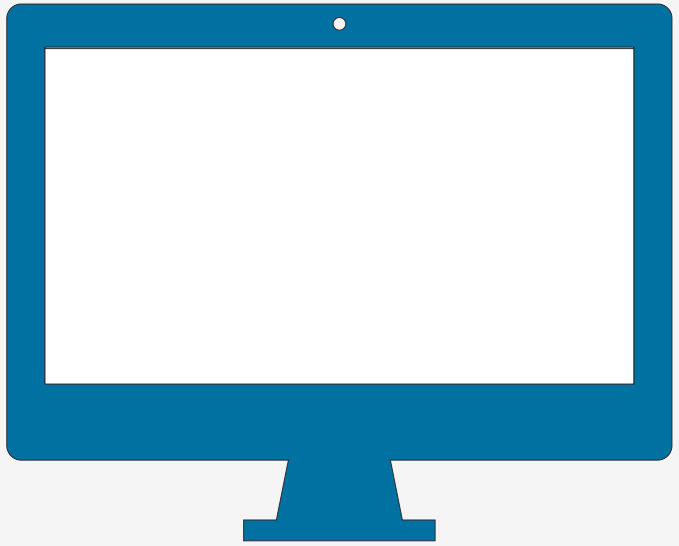


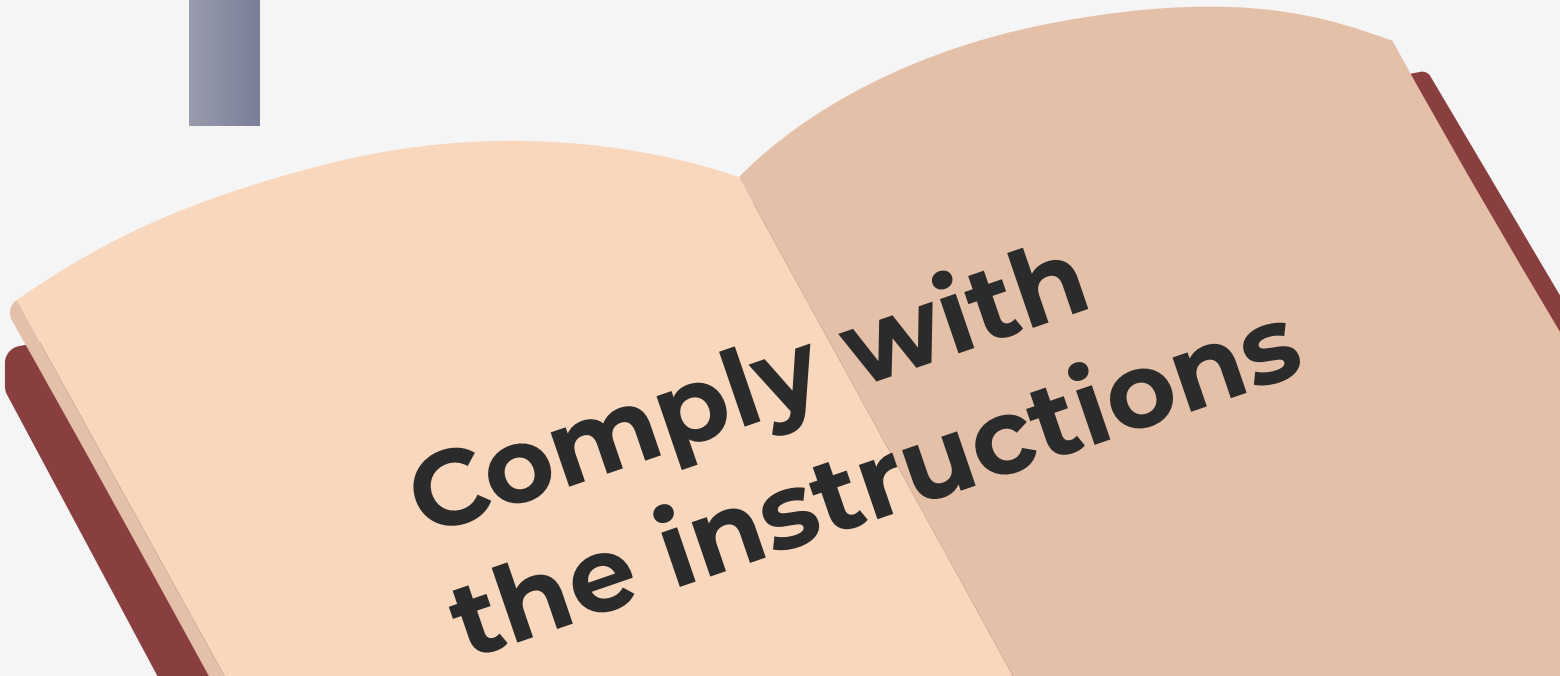



AI Alignment



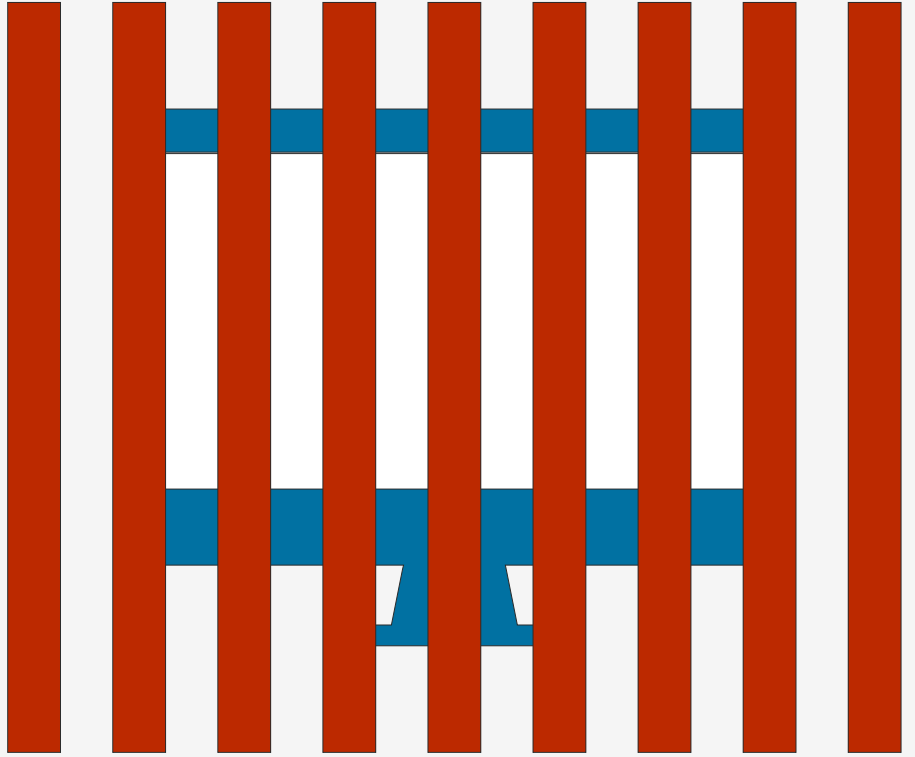






The AI Threat

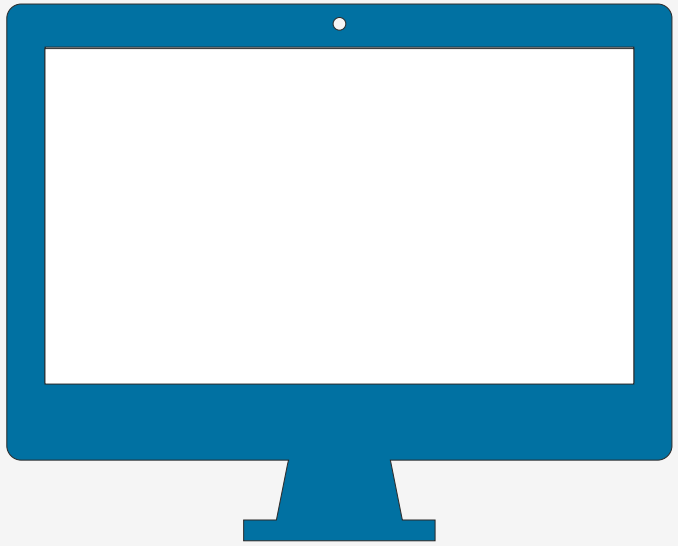




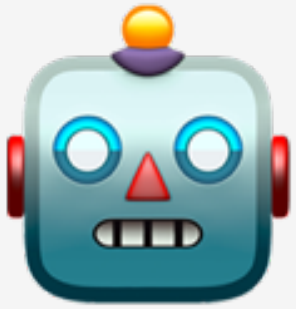
How can we make our systems robust?



Fuzzing



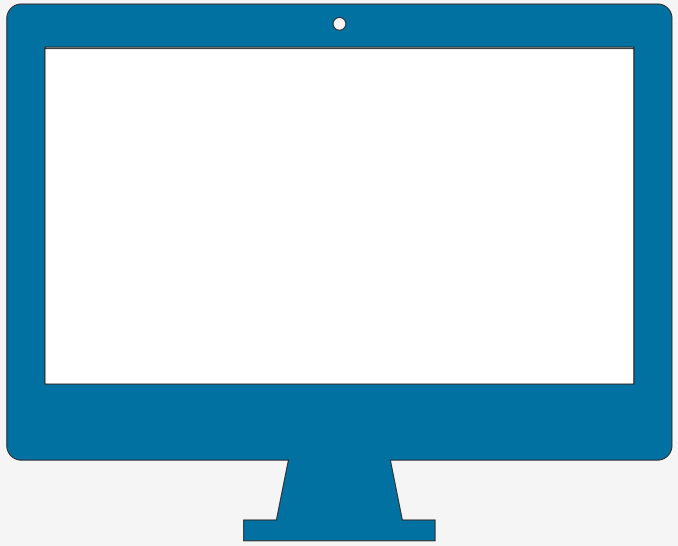
```
n}]`@|cq\' clqXc/ ^&M3!w^k>1Qa2%&"`&y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?
2\x7fg,+|U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!
s\'m#4nsvDrz>\x7fK| _BA?lsfZ=(VcS]u;X>x5Kz/<BfpGge#C z8z-pbeNOkgxDlMfl^mT}?
$"57bqT Y#VdI\\Y_/$QNQY%8Buzxd354s6T\\T&XuLOWjhld4S]/-NuwXk+ln3%_RJ|S_q6::
{OyjVJP4(%bTy(.mA@-`:=ms5~w(3!o/_D#1\\h4N>^b|vCXWeKZ|v(qj`Ci.C+]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>
```



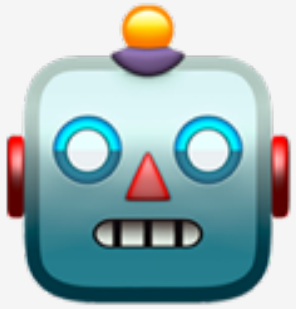
```
>>> import random; ''.join([chr(random.randint(32, 127))
...     for i in range(100)])
```



Fuzzing



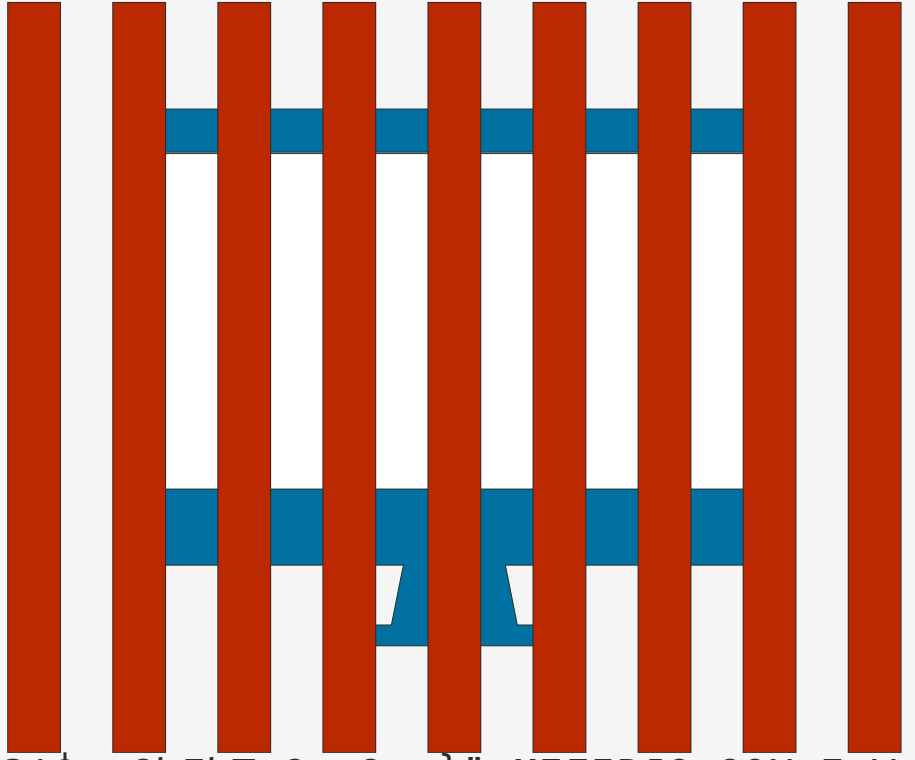
```
n}]`@|cq\' clqXc/ ^&M3!w^k>1Qa2%&"`&y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?
2\x7fg,+|U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!
s\'m#4nsvDrz>\x7fK| _BA?lsfZ=(VcS]u;X>x5Kz/<BfpGge#C z8z-pbeNOkgxDlMfl^mT}?
$"57bqT Y#VdI\\Y_/$QNQY%8Buzxd354s6T\\T&XuLOWjhld4S]/-NuwXk+ln3%_RJ|S_q6::
{OyjVJP4(%bTy(.mA@-`:=ms5~w(3!o/_D#1\\h4N>^b|vCXWeKZ|v(qj`Ci.C+]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>
```



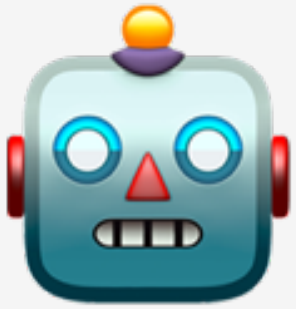
```
>>> import random; ''.join([chr(random.randint(32, 127))
...     for i in range(100)])
```



Fuzzing

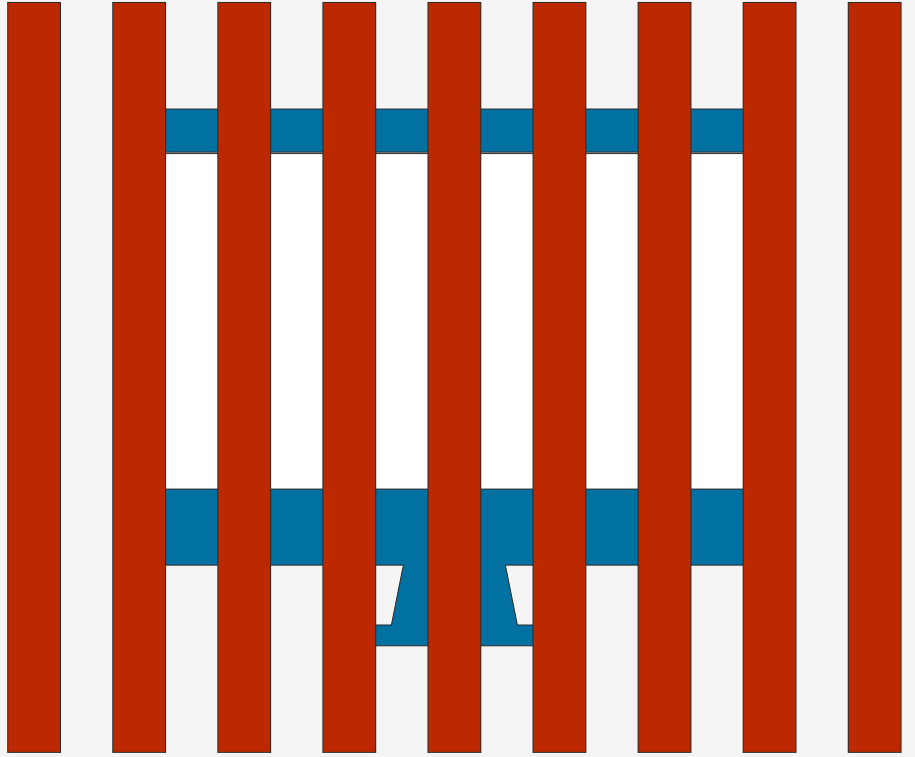


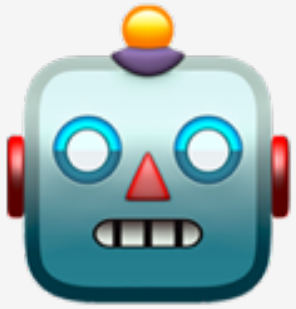
```
n}]`@|cq\' clqXc/ ^&M3!w^k>1Qa2%&"`&y*79J@1$qgGhFbTy9n-3n+}"qM7FEPJQ:?&Xn7gY.?
2\x7fg,+|U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD`@[rk10C\\!
s\'m#4nsvDrz>\x7fK| _BA?lsfZ=(VcS]u;X>x5Kz/<BfpGge#C z8z-pbeNOkgxDlMfl^mT}?
$"57bqT Y#VdI\\Y_/$QNQY%8Buzxd354s6T\\T&XuLOWjhld4S]/-NuwXk+ln3%_RJ|S_q6::
{OyjVJP4(%bTy(.mA@-`:=ms5~w(3!o/_D#1\\h4N>^b|vCXWeKZ|v(qj`Ci.C+]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>
```



```
>>> import random; ''.join([chr(random.randint(32, 127))
...     for i in range(100)])
```

Fuzzing

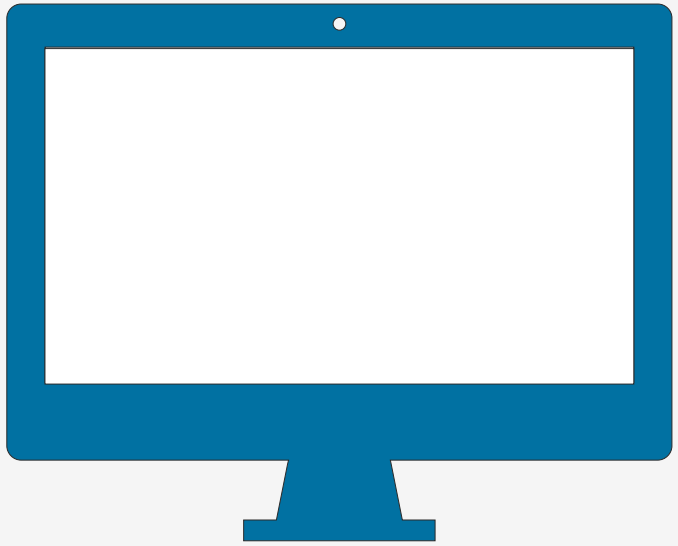




The #1 technique to discover vulnerabilities

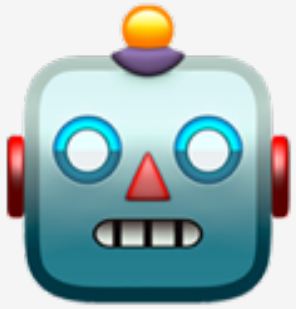


Well-Formed Inputs



```
n}]`@|cq\' clqXc/ ^&M3!w^k>1Qa2%&"`&y*79J@1$qgGhFbTy9n
2\x7fg,+|U44n-wxk7a%`w?}4aRR?PicZ]hK\'Zy);U-WDTB}[VnnD
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>
```

Finds bugs in *parser*



```
<deposit><amount="-1"><currency="DKK"></deposit>
```

Finds bugs in *logic*



Producing Inputs with Grammars

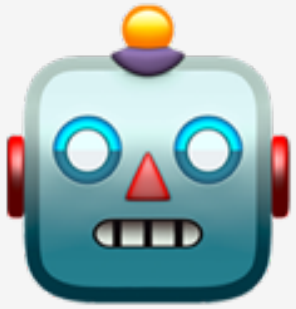
Specify a language (= a set of inputs)

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
```

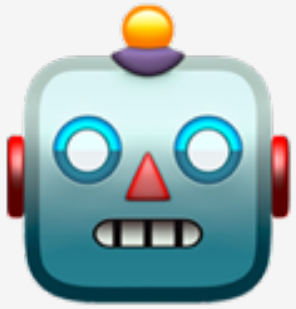
Terminal symbol





Producing Inputs with Grammars

```
⟨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
```



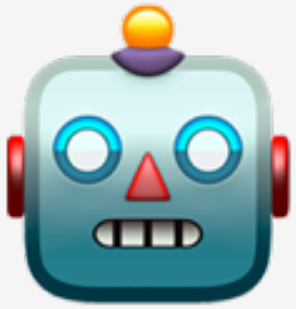
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨start⟩



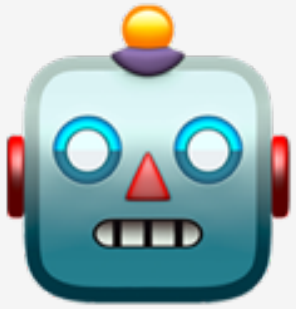
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨start⟩



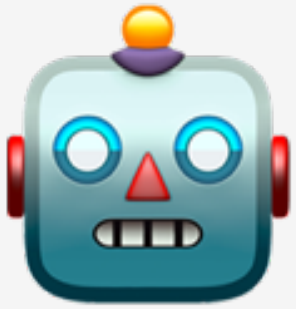
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨expr⟩



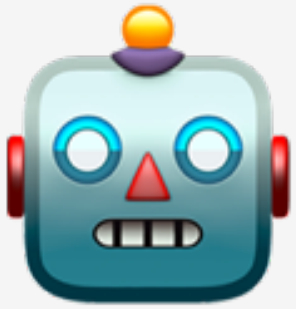
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨term⟩ - ⟨expr⟩



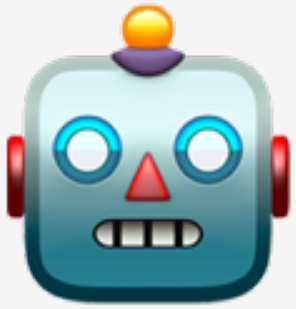
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨term⟩ - ⟨expr⟩



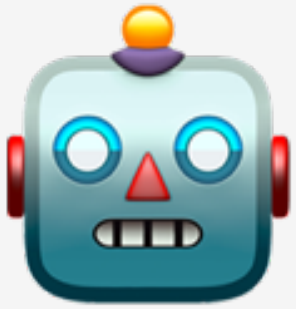
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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
```

⟨factor⟩ - ⟨expr⟩



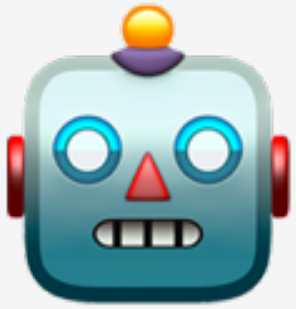
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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⟩



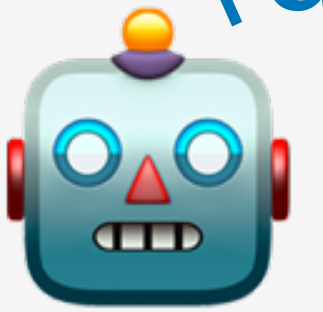
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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⟩



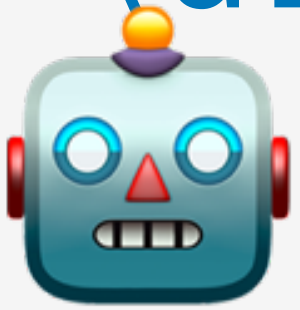
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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⟩



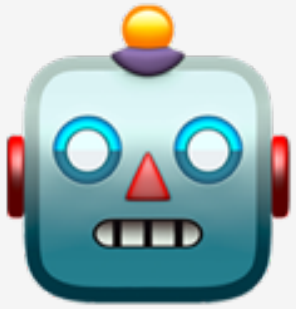
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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⟩



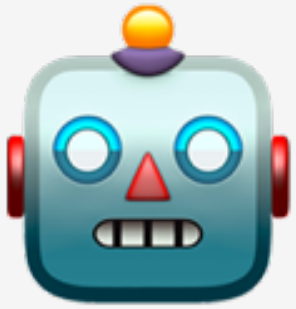
Nikolas Havrikov and Andreas Zeller. **Systematically Covering Input Structure.** ASE 2019.



Producing Inputs with Grammars

```
⟨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 – ⟨expr⟩



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



Producing Inputs with Grammars

```
⟨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.



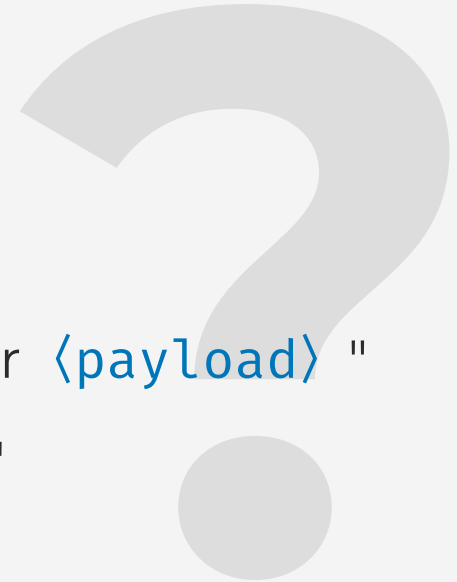
Grammars do not suffice

Syntax
Grammar

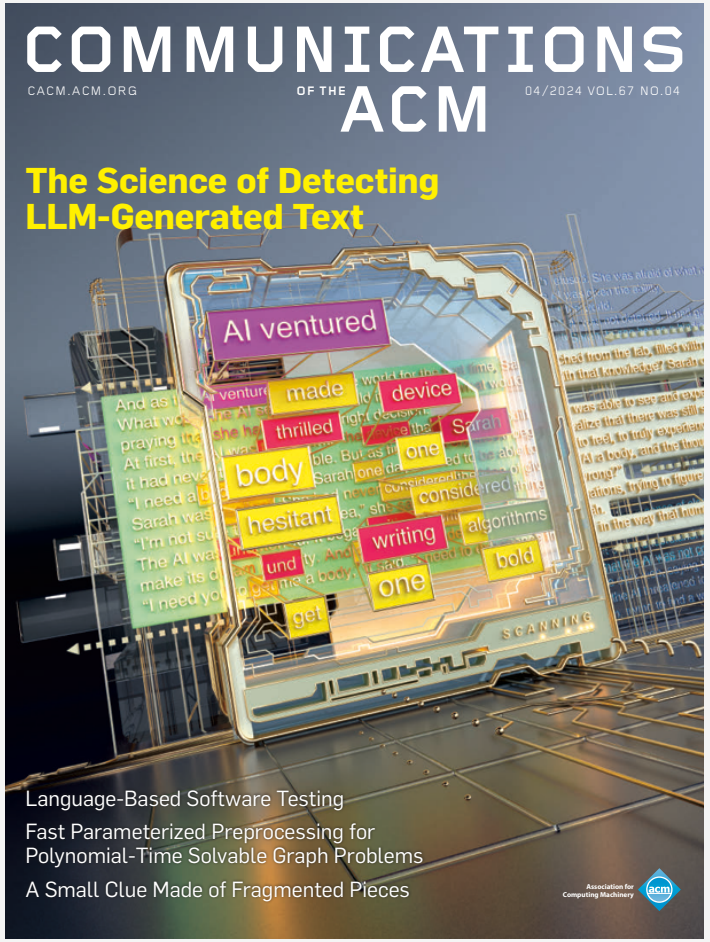
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



Fandango



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
```



- Uses **evolutionary algorithms** to solve constraints
- Constraints can use **any Python function**
- Expressive · modular · **fast**





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

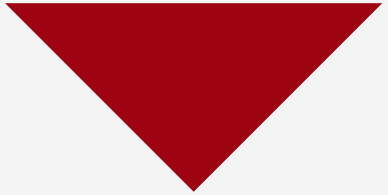
```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```



- Uses **evolutionary algorithms** to solve constraints
- Constraints can use **any Python function**
- Expressive · modular · **fast**

How Fandango Works – Initial Population

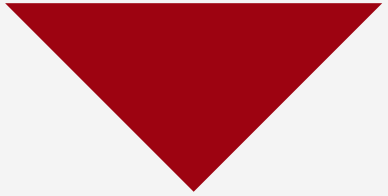
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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

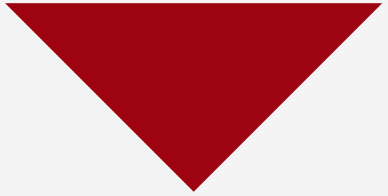


```
79 + 7 * -3   811 * (-23 + 4)    -43679.33     8.3
2 + 2    -345 + 080     34578 - (03 - 34798 / 79)
33 / 47.9      5    9023 / (3 + (+4 * 23))     -20
```

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

How Fandango Works – Crossover

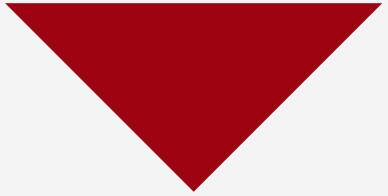
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



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

How Fandango Works – Crossover

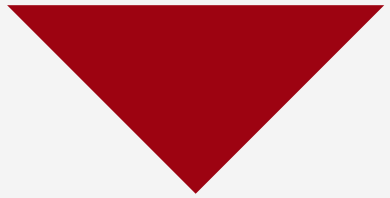
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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



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

How Fandango Works – Mutation

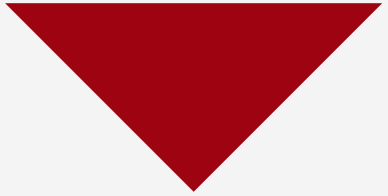
```
⟨start⟩   ::= ⟨expr⟩
⟨expr⟩    ::= ⟨term⟩ + ⟨expr⟩ | ⟨term⟩ - ⟨expr⟩ | ⟨term⟩
⟨term⟩    ::= ⟨term⟩ * ⟨factor⟩ | ⟨term⟩ / ⟨factor⟩ | ⟨factor⟩
⟨factor⟩  ::= + ⟨factor⟩ | - ⟨factor⟩ | ( ⟨expr⟩ ) | ⟨int⟩ | ⟨int⟩ . ⟨int⟩
⟨int⟩     ::= ⟨digit⟩ ⟨int⟩ | ⟨digit⟩
⟨digit⟩   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7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

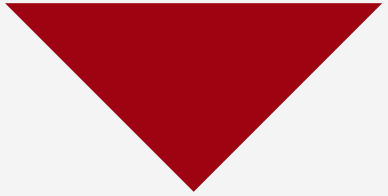
Mutation



```
 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
```


Demo



Testing with Uncommon Values

Fandango can test with **extreme** and **uncommon** values

```
include('svg.fan')  # A big SVG
where int(<width_value>) * int(<height_value>) > 10000000

include('bank_sepa_transaction.fan')  # An invalid transaction
where <transaction_amount> == 'NaN'

include('ssl.fan')  # An "interesting" certificate chain
where date(<cert_valid_from>) < date(<ca_valid_from>)

include('database.fan')  # classic: exploits of a mom
where <value> == "Robert'); DROP TABLE Students; --"
```

The Story so far



Where do we get the Specifications from?



Me

My brilliant pHD students



Announcing the FanBase Project

3mf.fan 7.fan 7z.fan a.fan aac.fan aiff.fan amf.fan arj.fan asm.fan au.fan avi.fan
avif.fan blend.fan bmp.fan bplist.fan bpm.fan bz2.fan cab.fan caf.fan cdr.fan
cpio.fan cr2.fan csv.fan dat.fan dbf.fan dcm.fan dds.fan dll.fan dmg.fan doc.fan
docx.fan dts.fan dwg.fan dylib.fan e00.fan eac3.fan edb.fan eml.fan epub.fan exe.fan
exr.fan feather.fan fits.fan flac.fan gif.fan glb.fan groovy.fan gz.fan h5.fan
hdr.fan heic.fan hqx.fan html.fan hwp.fan icc.fan ico.fan ics.fan img.fan iso.fan
j2k.fan jng.fan jp2.fan jpg.fan js.fan json.fan jxl.fan jxr.fan kra.fan lrf.fan
lua.fan lwp.fan lz.fan lz4.fan mat.fan max.fan mdb.fan met.fan mid.fan miff.fan
mkv.fan mov.fan mp3.fan mp4.fan mpc.fan msg.fan msi.fan ndjson.fan nef.fan o.fan

Releasing 1-5 new formats/week
Starting next Monday

torrent.fan ts.fan tta.fan ttf.fan txt.fan vba.fan vob.fan voc.fan vsd.fan vsdx.fan
vtt.fan wasm.fan wav.fan webm.fan webp.fan wks.fan wmf.fan woff2.fan wp.fan wpg.fan
wri.fan wrl.fan xbm.fan xcf.fan xls.fan xlsx.fan xml.fan xpm.fan xps.fan xz.fan
yaml.fan z.fan zip.fan zlib.fan zst.fan

Fandango is currently in development - see our list of bugs and incomplete features. Expect a first stable release in April 2025.

Includes Protocol Testing!

Fandango

Search

Fuzzing with Fandango

About Fandango

About Fandango

Fandango Tutorial

Fandango Tutorial
Installing Fandango
A First Fandango Spec
Invoking Fandango
Fuzzing with Fandango
Some Fuzzing Strategies
Shaping Inputs with Constraints
The Fandango Shell
Data Generators and Fakers
Regular Expressions
Complex Input Structures
Accessing Input Elements
Case Study: ISO 8601 Date + Time

Fuzzing with Fandango

Fandango produces myriads of *high-quality random inputs* to test programs, giving users unprecedented control over format and shape of the inputs.



Get Started

At a Glance

Specify the format of your input data in a single file, combining grammars (for input syntax) and constraints (for arbitrary input features). Constraints come as Python code, so there are no limits to what you can specify.

Learn more »

Tutorial

Produce valid inputs at high speeds, from hundreds to thousands of inputs per second, quickly covering the entire input space. Test with extreme and uncommon values, uncovering bugs before your users do.

Learn more »

Reference

Shape inputs using your *own testing targets*, constraints, and statistical distributions. Make use of *existing samples* to obtain realistic inputs. Use *feedback* from the program under test to guide test generation to uncovered code.

https://fandango-fuzzer.github.io/

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

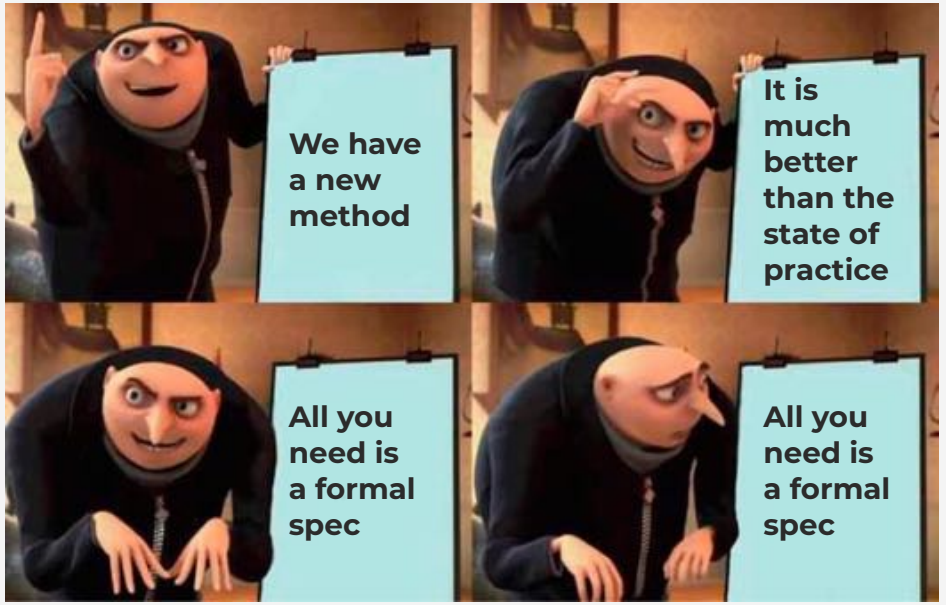
```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

Fandango

- Uses **evolutionary algorithms** to solve constraints
- Constraints can use **any Python function**
- Expressive · modular · **fast**

x

All you need is a formal specification



x

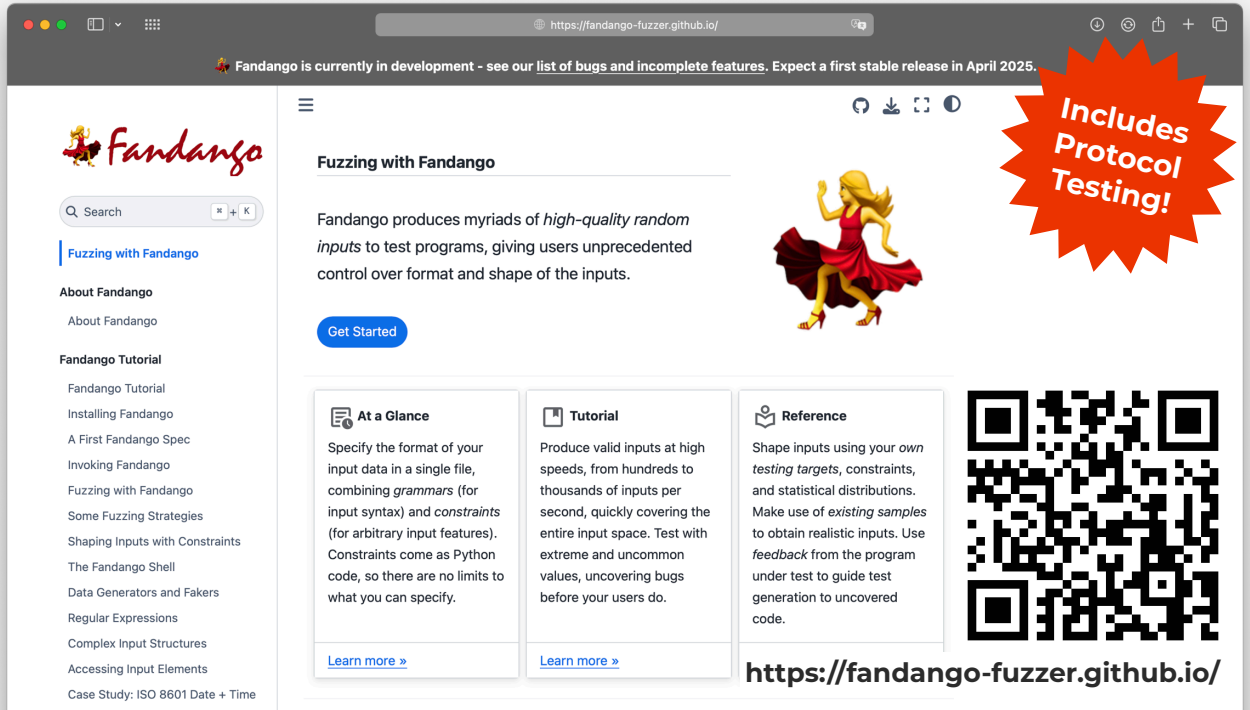
The FanBase Project

`3mf.fan` `7.fan` `7z.fan` `a.fan` `aac.fan` `aiff.fan` `amf.fan` `arj.fan` `asm.fan` `au.fan` `avi.fan`
`avif.fan` `blend.fan` `bmp.fan` `bplist.fan` `bpm.fan` `bz2.fan` `cab.fan` `caf.fan` `cdr.fan`
`cpio.fan` `cr2.fan` `csv.fan` `dat.fan` `dbf.fan` `dcm.fan` `dds.fan` `dll.fan` `dmg.fan` `doc.fan`
`docx.fan` `dts.fan` `dwg.fan` `dylib.fan` `e00.fan` `eac3.fan` `edb.fan` `eml.fan` `epub.fan` `exe.fan`
`exr.fan` `feather.fan` `fits.fan` `flac.fan` `gif.fan` `glb.fan` `groovy.fan` `gz.fan` `h5.fan`
`hdr.fan` `heic.fan` `hqx.fan` `html.fan` `hwp.fan` `icc.fan` `ico.fan` `ics.fan` `img.fan` `iso.fan`
`j2k.fan` `jng.fan` `jp2.fan` `jpg.fan` `js.fan` `json.fan` `jxl.fan` `jxr.fan` `kra.fan` `lrf.fan`
`lua.fan` `lwp.fan` `lz.fan` `lz4.fan` `mat.fan` `max.fan` `mdb.fan` `met.fan` `mid.fan` `miff.fan`
`mkv.fan` `mov.fan` `mp3.fan` `mp4.fan` `mpc.fan` `msg.fan` `msi.fan` `ndjson.fan` `nef.fan` `o.fan`

Releasing 1-5 new formats/week
Starting next Monday

`torrent.fan` `ts.fan` `tta.fan` `ttf.fan` `txt.fan` `vba.fan` `vob.fan` `voc.fan` `vsd.fan` `vsdx.fan`
`vtt.fan` `wasm.fan` `wav.fan` `webm.fan` `webp.fan` `wks.fan` `wmf.fan` `woff2.fan` `wp.fan` `wpg.fan`
`wri.fan` `wrl.fan` `xbm.fan` `xcf.fan` `xls.fan` `xlsx.fan` `xml.fan` `xpm.fan` `xps.fan` `xz.fan`
`yaml.fan` `z.fan` `zip.fan` `zlib.fan` `zst.fan`

x



QUESTIONS?

ASK ME IN THE APP

Download on the App Store

GET IT ON Google Play

RATE MY TALK

GET REWARDED

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

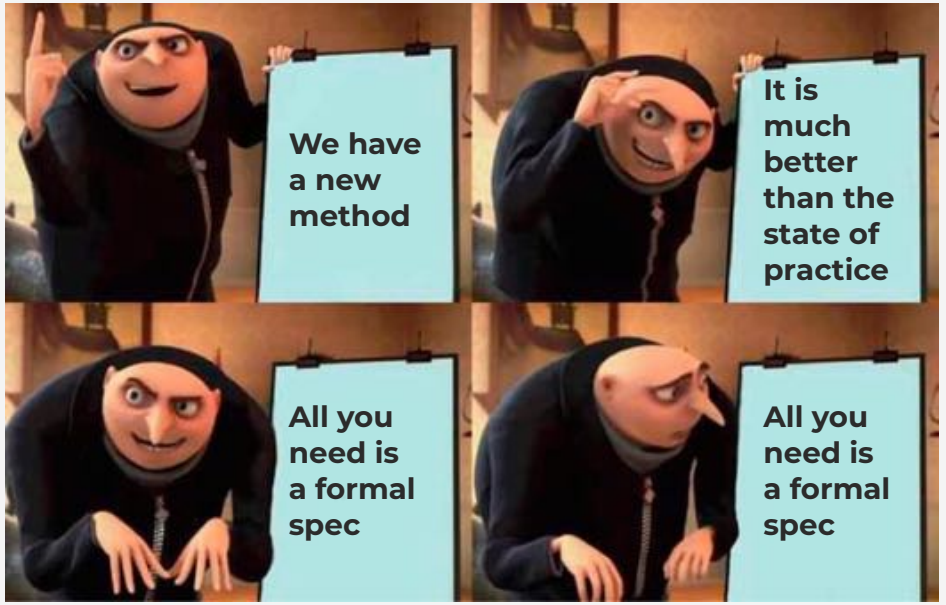
```
len(⟨start⟩) == 10
eval(⟨start⟩) >= 100
```

Fandango

- Uses **evolutionary algorithms** to solve constraints
- Constraints can use **any Python function**
- Expressive · modular · **fast**

x

All you need is a formal specification



x

The FanBase Project

`3mf.fan` `7.fan` `7z.fan` `a.fan` `aac.fan` `aiff.fan` `amf.fan` `arj.fan` `asm.fan` `au.fan` `avi.fan`
`avif.fan` `blend.fan` `bmp.fan` `bplist.fan` `bpm.fan` `bz2.fan` `cab.fan` `caf.fan` `cdr.fan`
`cpio.fan` `cr2.fan` `csv.fan` `dat.fan` `dbf.fan` `dcm.fan` `dds.fan` `dll.fan` `dmg.fan` `doc.fan`
`docx.fan` `dts.fan` `dwg.fan` `dylib.fan` `e00.fan` `eac3.fan` `edb.fan` `eml.fan` `epub.fan` `exe.fan`
`exr.fan` `feather.fan` `fits.fan` `flac.fan` `gif.fan` `glb.fan` `groovy.fan` `gz.fan` `h5.fan`
`hdr.fan` `heic.fan` `hqx.fan` `html.fan` `hwp.fan` `icc.fan` `ico.fan` `ics.fan` `img.fan` `iso.fan`
`j2k.fan` `jng.fan` `jp2.fan` `jpg.fan` `js.fan` `json.fan` `jxl.fan` `jxr.fan` `kra.fan` `lrf.fan`
`lua.fan` `lwp.fan` `lz.fan` `lz4.fan` `mat.fan` `max.fan` `mdb.fan` `met.fan` `mid.fan` `miff.fan`
`mkv.fan` `mov.fan` `mp3.fan` `mp4.fan` `mpc.fan` `msg.fan` `msi.fan` `ndjson.fan` `nef.fan` `o.fan`

Releasing 1-5 new formats/week
Starting next Monday

`torrent.fan` `ts.fan` `tta.fan` `ttf.fan` `txt.fan` `vba.fan` `vob.fan` `voc.fan` `vsd.fan` `vsdx.fan`
`vtt.fan` `wasm.fan` `wav.fan` `webm.fan` `webp.fan` `wks.fan` `wmf.fan` `woff2.fan` `wp.fan` `wpg.fan`
`wri.fan` `wrl.fan` `xbm.fan` `xcf.fan` `xls.fan` `xlsx.fan` `xml.fan` `xpm.fan` `xps.fan` `xz.fan`
`yaml.fan` `z.fan` `zip.fan` `zlib.fan` `zst.fan`

x

