# Taming Fuzzers

FuzzCon Europe • September 8, 2020

Andreas Zeller

with Rahul Gopinath, Rafael Dutra, and Zeller's team at CISPA

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

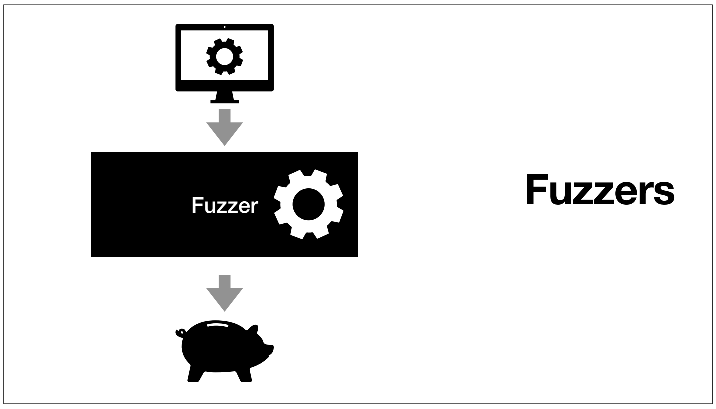
**Taming Fuzzers**

Andreas Zeller, CISPA Helmholtz Center for Information Security
Joint work with Rahul Gopinath, Rafael Dutra, and Zeller's team at CISPA

Software test generation (fuzzing) can be made much more effective if one knows what to search for. But how can users inform fuzzers about the program and its domain? And how can they control what a fuzzer should do?

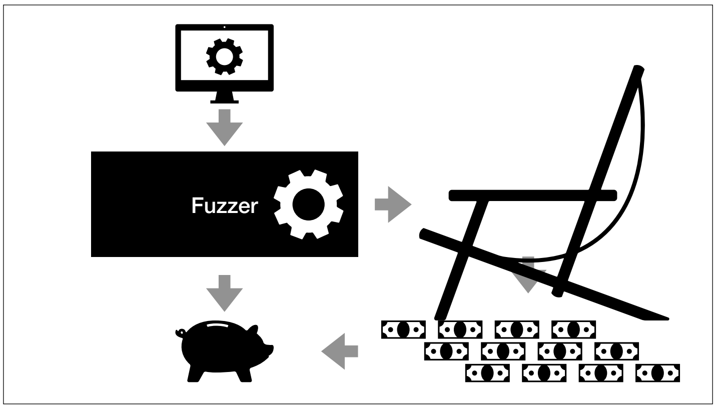
In this talk, I present and introduce tools and techniques that allow users to *specify* the languages of program inputs, from recursive languages such as JavaScript to complex binary inputs, leveraging hundreds of existing format specifications. And I show how to *customize* such languages, targeting specific input features. Our all-new FormatFuzzer is now available as open source.

https://andreas-zeller.info/
https://www.cispa.de/

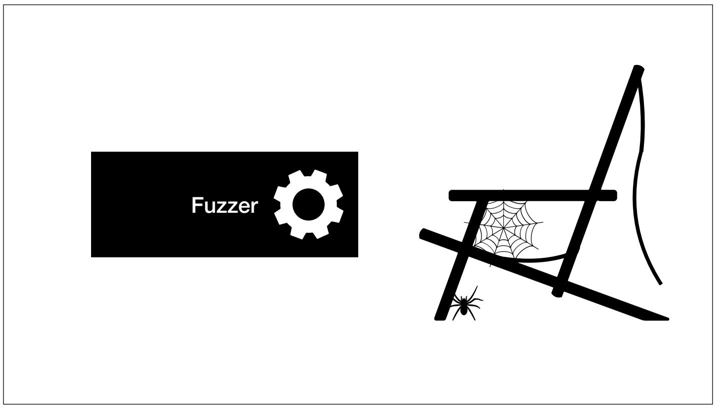


## Fuzzers

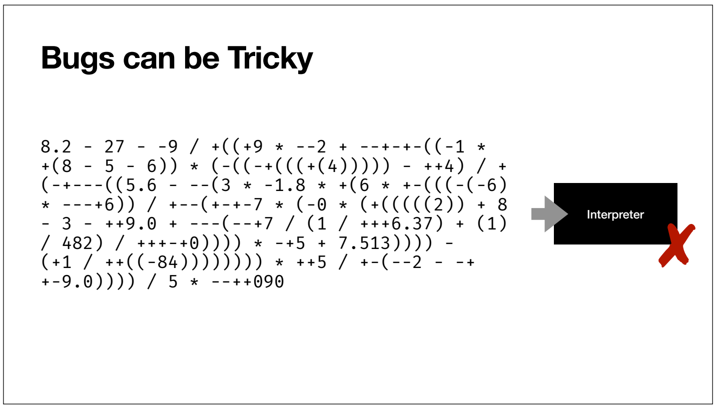
We imagine fuzzers as machines that take programs to produce bugs, and hence money.



... and, of course, all of this automatically.



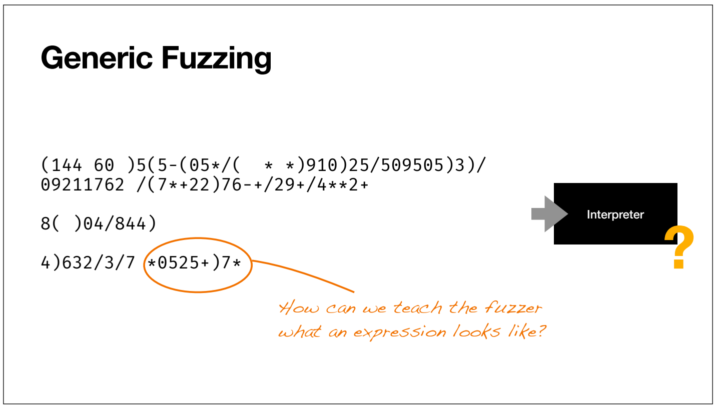
Unfortunately, this doesn't work. (Or takes a long time.)



## Bugs can be Tricky

```
8.2 - 27 - -9 / +((+9 * --2 + --+-+-((-1 *
+(8 - 5 - 6)) * (-((-+(((+(4)))))) - ++4) / +
(-+---((5.6 - --(3 * -1.8 * +(6 * +-(((-(-6)
* ---+6)) / +--(+-+-7 * (-0 * (+(((((2)) + 8
- 3 - ++9.0 + ---(--+7 / (1 / +++6.37) + (1)
/ 482) / +++-+0)))) * -+5 + 7.513)))) -
(+1 / ++((-84))))))))) * ++5 / +-(--2 - -+
+-9.0)))) / 5 * --++090
```

Because if you have a bug in a program with a complex input



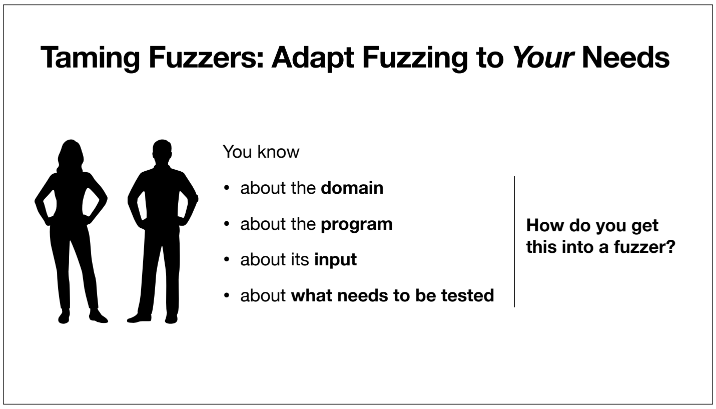
## Generic Fuzzing

```
(144 60 )5(5-(05*/(  * *)910)25/509505)3)/
09211762 /(7*+22)76-+/29+/4**2+

8(  )04/844)

4)632/3/7 *0525+)7*
```

Interpreter
?

How can we teach the fuzzer what an expression looks like?

... your fuzzer might not find them. If you just take sequences of random characters and throw them at an interpreter, all you're going to get is syntax errors. (It's okay to test syntax error handling, but this should not be all.)



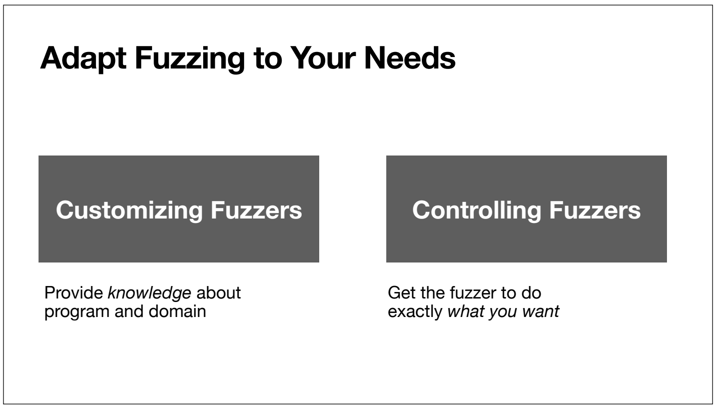
## Taming Fuzzers: Adapt Fuzzing to *Your* Needs

You know

- about the **domain**
- about the **program**
- about its **input**
- about **what needs to be tested**

**How do you get this into a fuzzer?**

So the theme of this talk is how to integrate user's knowledge into a fuzzer – to test more efficiently, and to test more effectively.



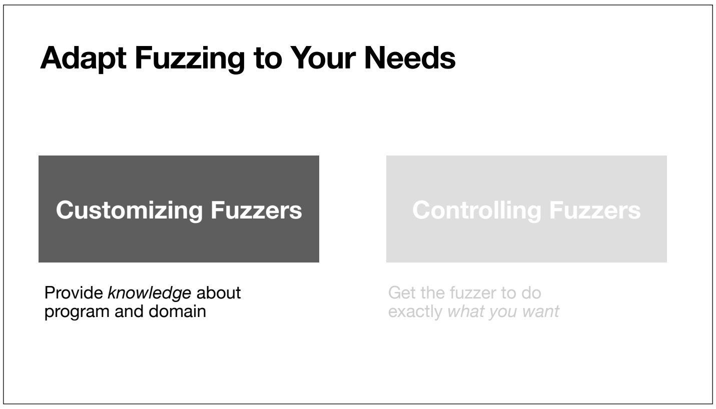
## Adapt Fuzzing to Your Needs

**Customizing Fuzzers**

Provide *knowledge* about program and domain

**Controlling Fuzzers**

Get the fuzzer to do exactly *what you want*



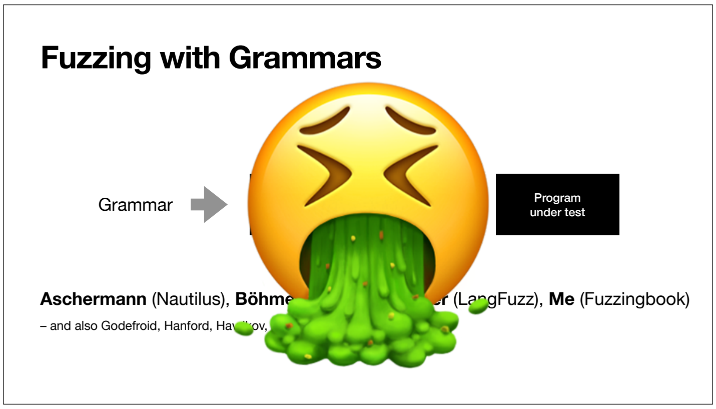
## Adapt Fuzzing to Your Needs

**Customizing Fuzzers**

Provide *knowledge* about program and domain

**Controlling Fuzzers**

Get the fuzzer to do exactly *what you want*

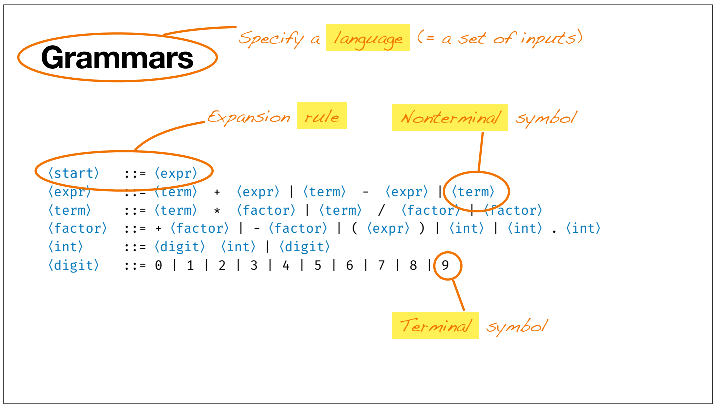


## Fuzzing with Grammars

**Aschermann** (Nautilus), **Böhme** **r** (LangFuzz), **Me** (Fuzzingbook)

– and also Godefroid, Hanford, Ha ov,

Grammars are popular tools in fuzzers – but not everyone likes the word "grammar".



## Grammars

Specify a language (= a set of inputs)

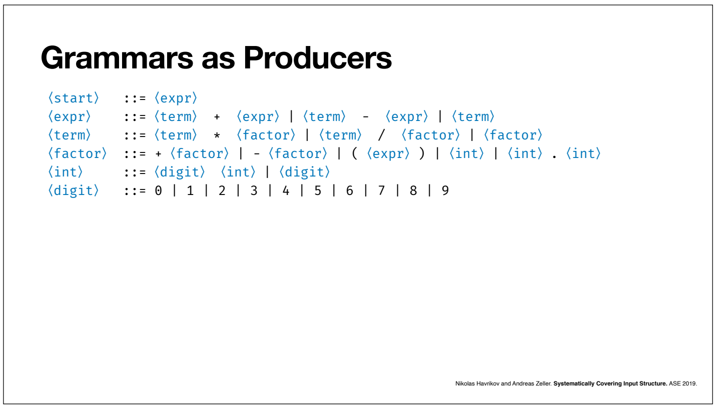
Expansion rule

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

I'll therefore stick to the most basic definitions.

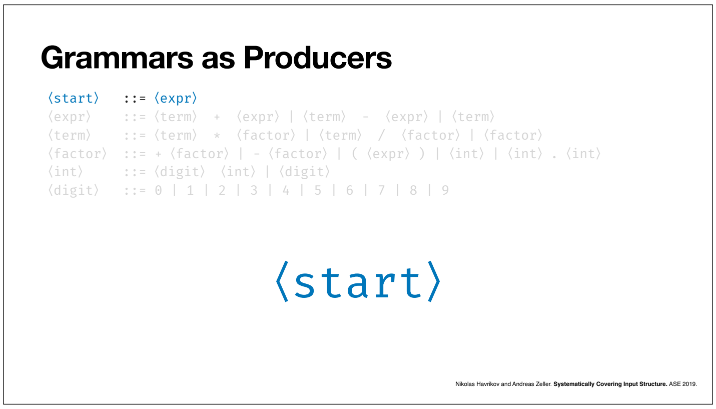


### Grammars as Producers

```
⟨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
```

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

You may have seen grammars as **parsers**, but they can also be used as **producers** of inputs.



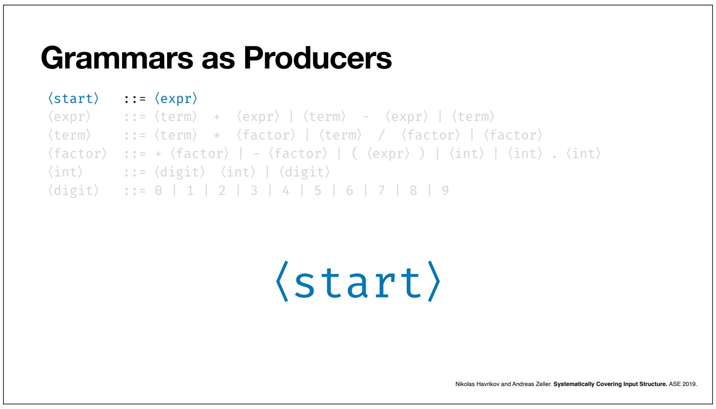
### Grammars as Producers

```
⟨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
```

⟨start⟩

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

You start with a start symbol

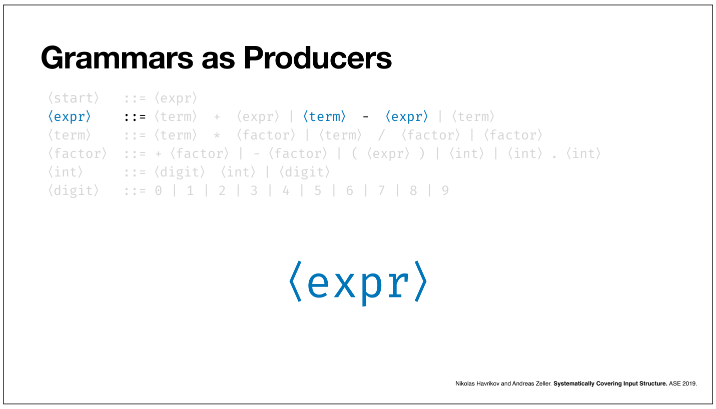


### Grammars as Producers

```
⟨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
```

⟨start⟩

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



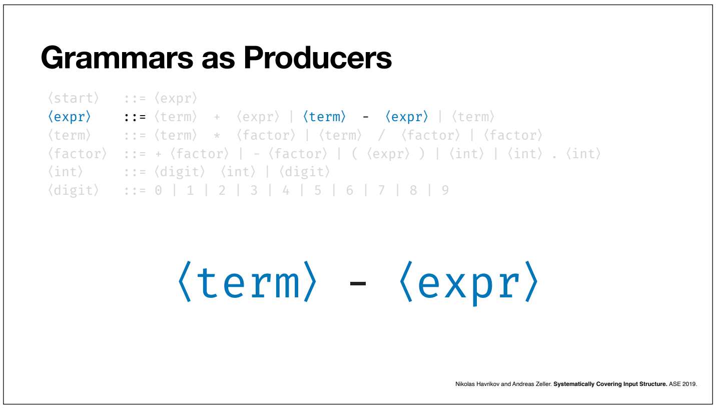
## Grammars as Producers

```
⟨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
```

⟨expr⟩

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

which then subsequently gets replaced according to the production rules in the grammar.



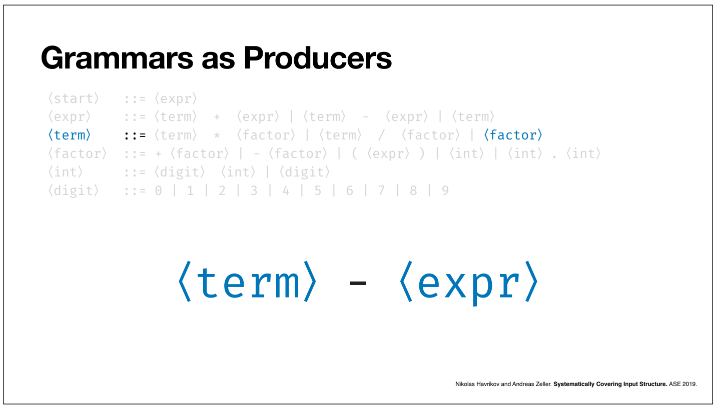
## Grammars as Producers

```
⟨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
```

⟨term⟩ - ⟨expr⟩

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

If there are multiple alternatives, you randomly choose one.

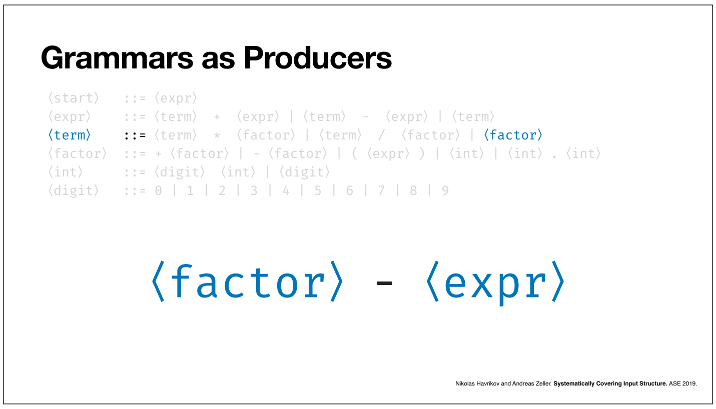


## Grammars as Producers

```
⟨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
```

⟨term⟩ - ⟨expr⟩

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

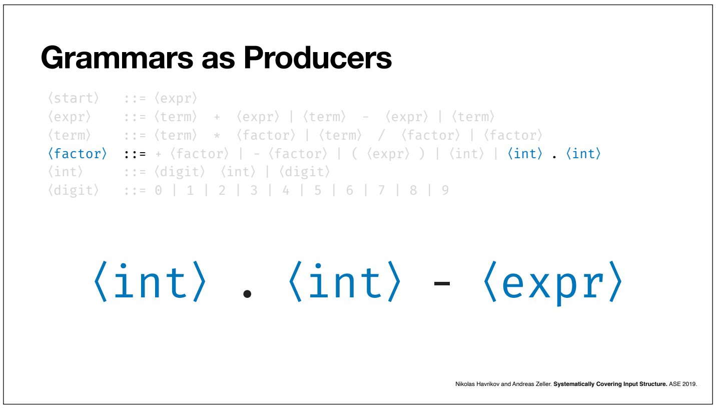


## Grammars as Producers

```
⟨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
```

```
⟨factor⟩ - ⟨expr⟩
```

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

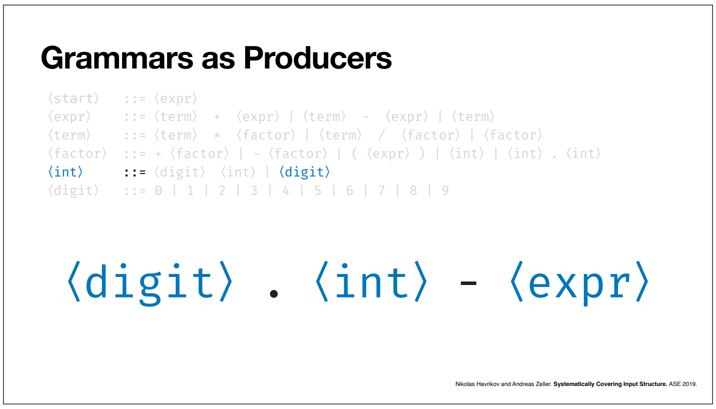


## Grammars as Producers

```
⟨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⟩
```

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

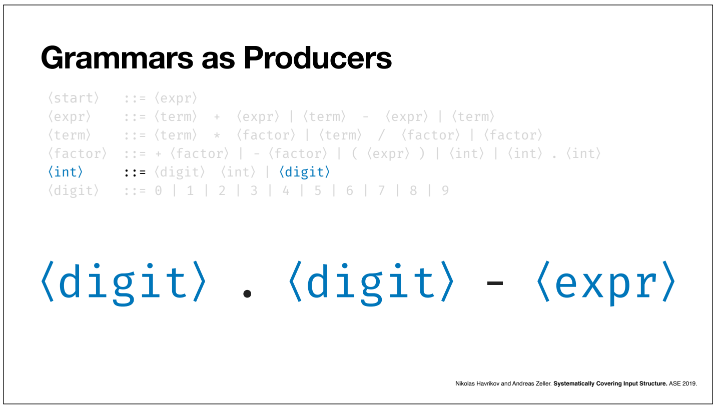


## Grammars as Producers

```
⟨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⟩
```

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

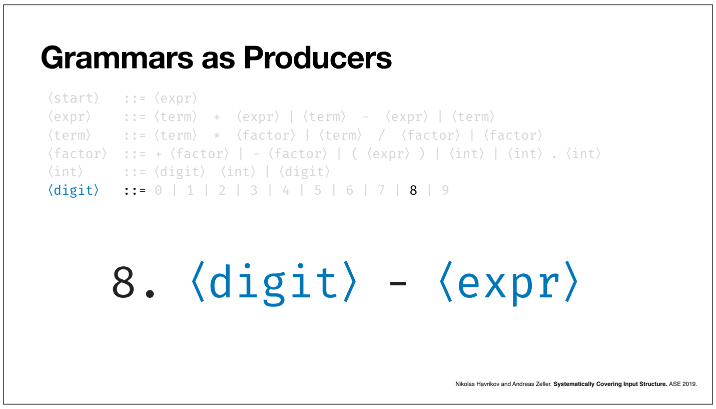


## Grammars as Producers

```
⟨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⟩
```

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

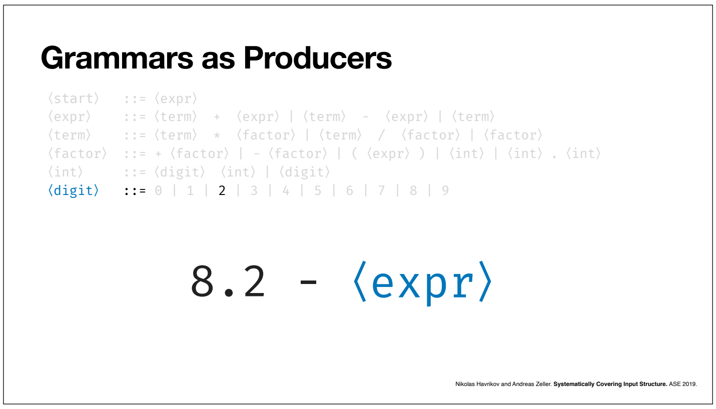


## Grammars as Producers

```
⟨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⟩
```

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



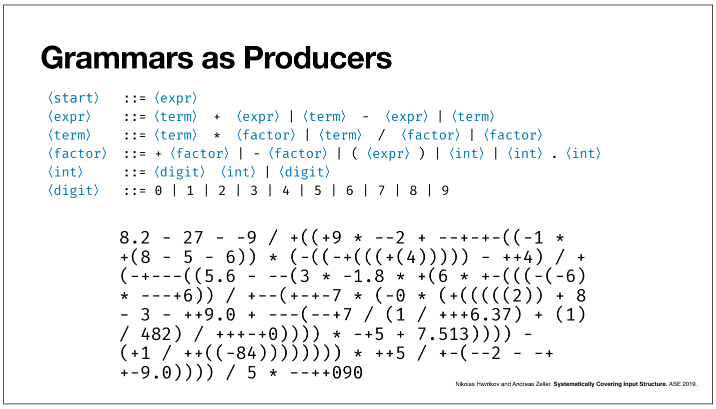
## Grammars as Producers

```
⟨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 - ⟨expr⟩
```

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

Over time, this gives you a syntactically valid input. In our case, a valid arithmetic expression.



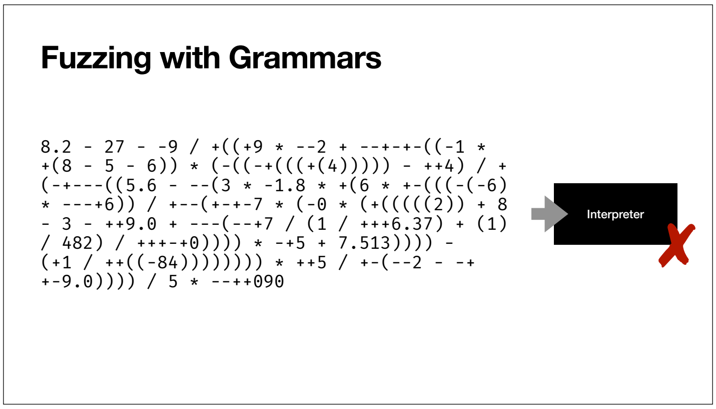
## Grammars as Producers

```
⟨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 - 27 - -9 / +((+9 * --2 + --+-+-((-1 *
+(8 - 5 - 6)) * (-((-+(((+(4)))))) - ++4) / +
(-+---((5.6 - --(3 * -1.8 * +(6 * +-(((-(-6)
* ---+6)) / +--(+-+-7 * (-0 * (+(((((2)) + 8
- 3 - ++9.0 + ---(--+7 / (1 / +++6.37) + (1)
/ 482) / +++-+0)))) * -+5 + 7.513)))) -
(+1 / ++((-84))))))))) * ++5 / +-(--2 - -+
+-9.0)))) / 5 * --++090
```

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

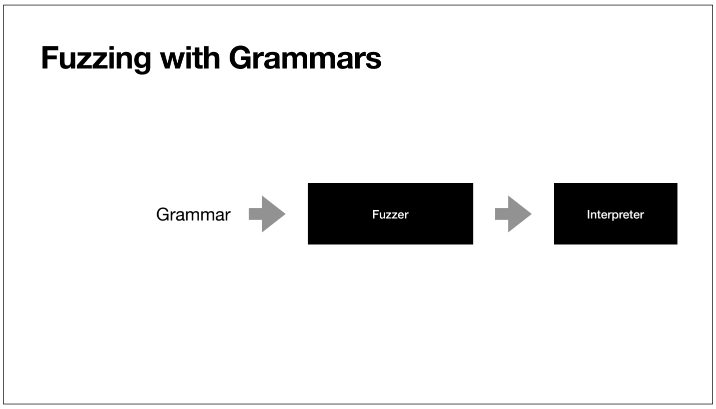
Reliably reachly, a pretty **complex** arithmetic expression.



## Fuzzing with Grammars

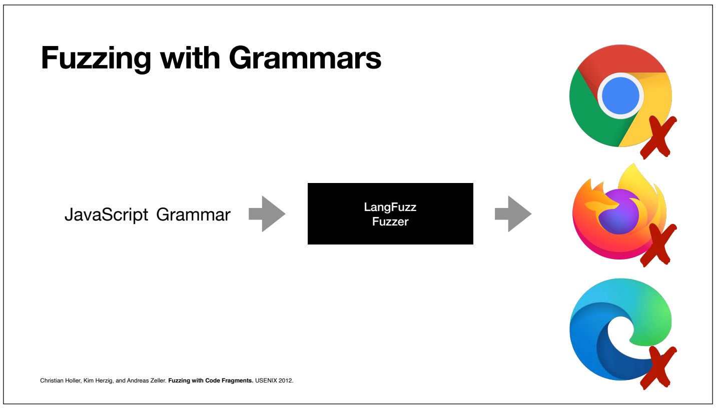
```
8.2 - 27 - -9 / +((+9 * --2 + --+-+-((-1 *
+(8 - 5 - 6)) * (-((-+(((+(4)))))) - ++4) / +
(-+---((5.6 - --(3 * -1.8 * +(6 * +-(((-(-6)
* ---+6)) / +--(+-+-7 * (-0 * (+(((((2)) + 8
- 3 - ++9.0 + ---(--+7 / (1 / +++6.37) + (1)
/ 482) / +++-+0)))) * -+5 + 7.513)))) -
(+1 / ++((-84))))))))) * ++5 / +-(--2 - -+
+-9.0)))) / 5 * --++090
```

These can now be used as input to your program. And by construction, all inputs are valid.



## Fuzzing with Grammars

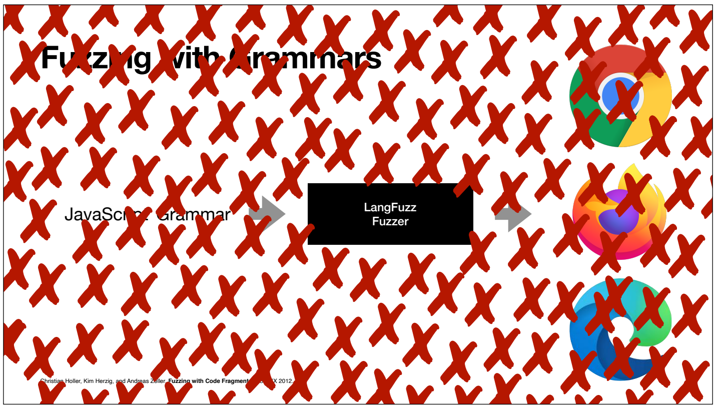
This actually scales.



## Fuzzing with Grammars

Christian Holler, Kim Herzig, and Andreas Zeller. **Fuzzing with Code Fragments.** USENIX 2012.

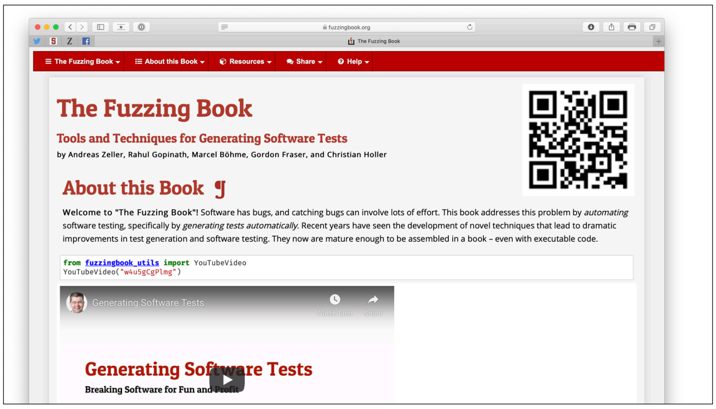
A couple of years ago, we used a JavaScript grammar to fuzz the interpreters of Firefox, Chrome and Edge.



Fuzzing with Grammars

Christian Holler, Kim Herzig, and Andreas Zeller. **Fuzzing with Code Fragments.** USENIX 2012.

My student Christian Holler (whom you've heard earlier today) found more than 2,600 bugs, and in the first four weeks, he netted more than $50,000 in bug bounties. If you use a browser to read this, one of the reasons your browser works as it should is because of grammar-based fuzzing.



# The Fuzzing Book

### Tools and Techniques for Generating Software Tests

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

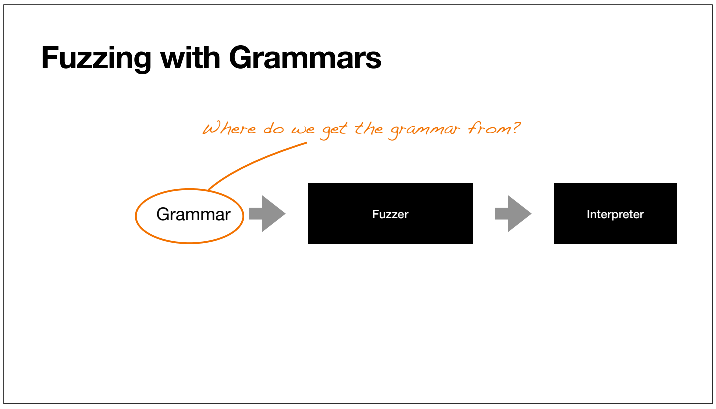
## About this Book

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

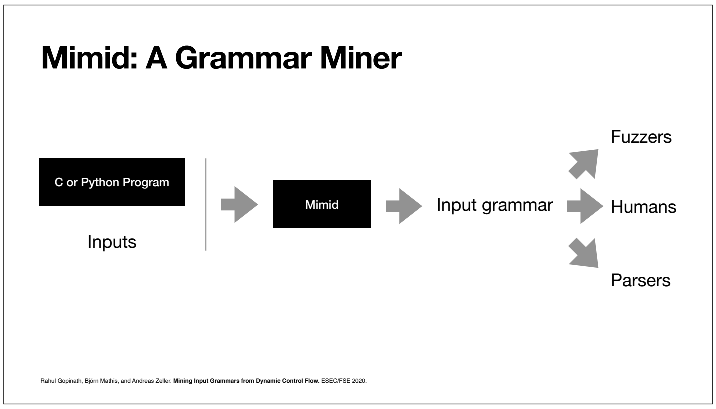
Generating Software Tests
Watch later
Share
Generating Software Tests
Breaking Software for Fun and Profit

If you want to learn more about these, have a look at our book – a textbook on fuzzing.



## Fuzzing with Grammars

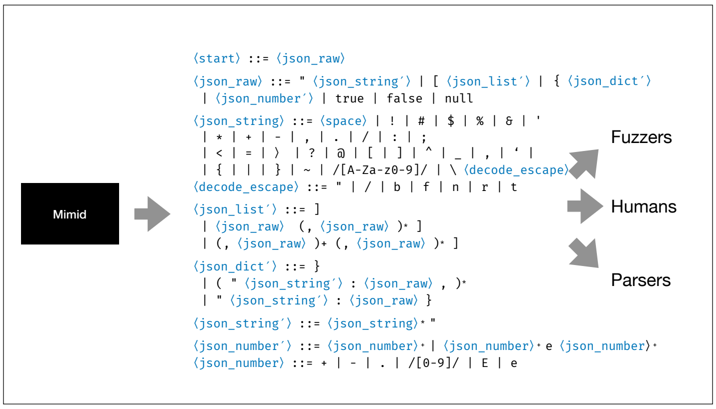
But the problem for all grammar-based approaches is: Where do you get this grammar from?



## Mimid: A Grammar Miner

Rahul Gopinath, Björn Mathis, and Andreas Zeller. **Mining Input Grammars from Dynamic Control Flow.** ESEC/FSE 2020.

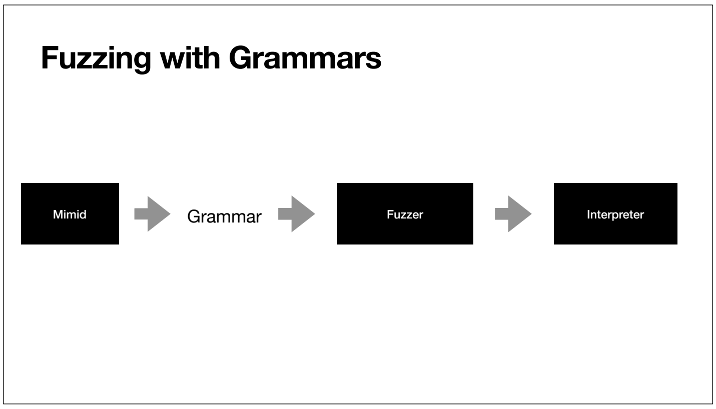
Our Mimid grammar miner takes a program and its inputs and extracts a grammar out of it. This grammar can directly be used by fuzzers, parsers, and humans.



```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | [ ⟨json_list′⟩ | { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | 〉 | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* "
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

Mimid
Fuzzers
Humans
Parsers

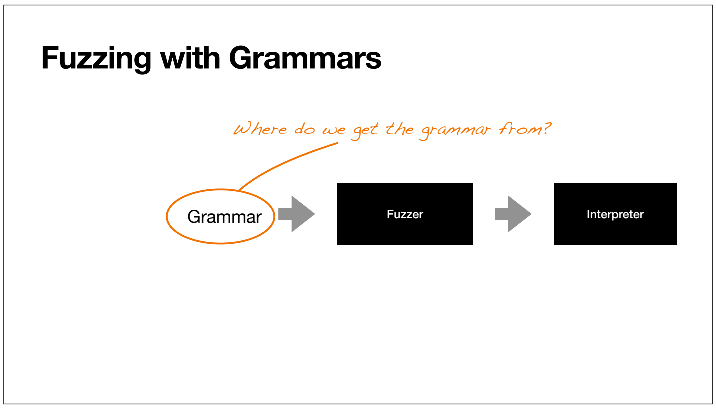
The extracted grammars are well structured and human readable as you can see in this grammar extracted from a JSON parser.



## Fuzzing with Grammars

Mimid → Grammar → Fuzzer → Interpreter

Our plan is that you can go, mine a grammar from a program, and then use this for highly effective fuzzing.

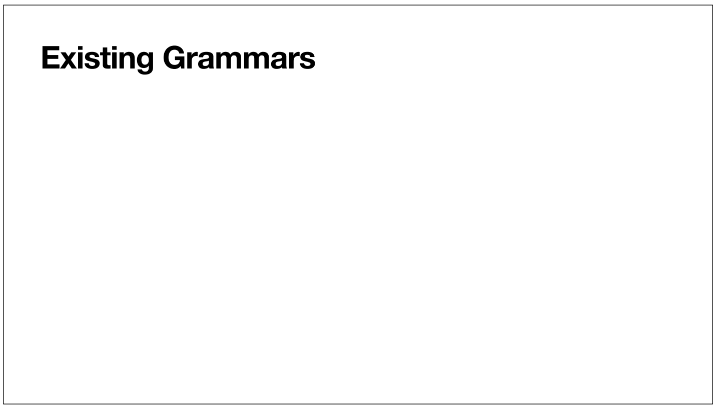


## Fuzzing with Grammars

Where do we get the grammar from?

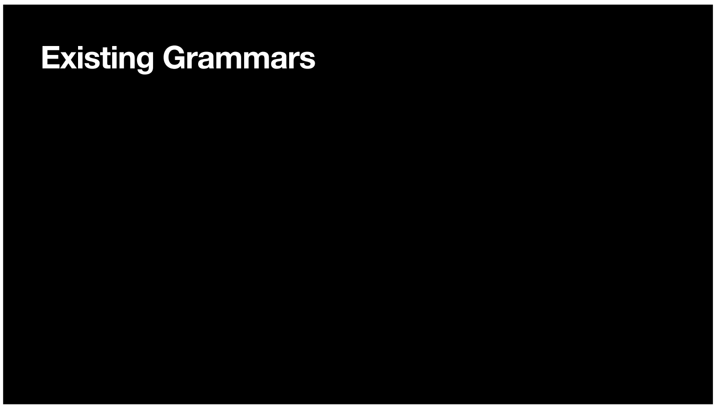
Grammar → Fuzzer → Interpreter

However, Mimid and grammar mining are still research prototypes – and also are limited to, well, grammars. What do you do for more complex inputs?



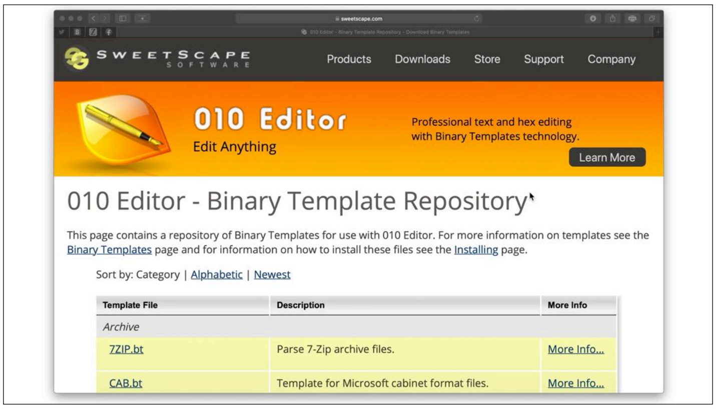
## Existing Grammars

You use an existing grammar!



## Existing Grammars

(Unfortunately, there are almost none that could be used for fuzzers – which made us investigate grammar mining in the first place.)



sweetscape.com
010 Editor - Binary Template Repository - Download Binary Templates
SWEETSCAPE SOFTWARE
Products
Downloads
Store
Support
Company
010 Editor
Edit Anything
Professional text and hex editing with Binary Templates technology.
Learn More

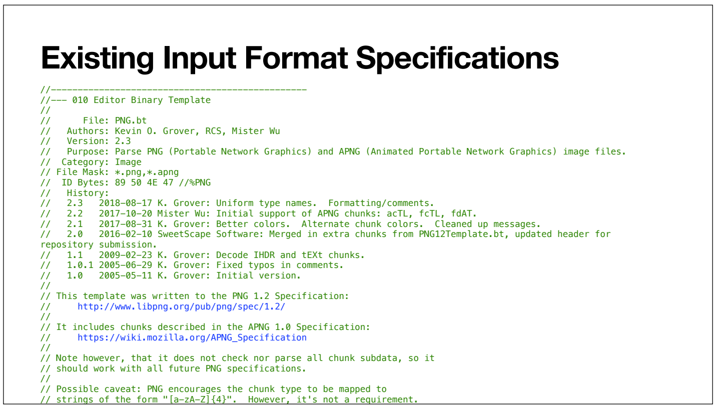
010 Editor - Binary Template Repository

This page contains a repository of Binary Templates for use with 010 Editor. For more information on templates see the Binary Templates page and for information on how to install these files see the Installing page.

Sort by: Category | Alphabetic | Newest

| Template File | Description | More Info |
|---|---|---|
| *Archive* | | |
| 7ZIP.bt | Parse 7-Zip archive files. | More Info... |
| CAB.bt | Template for Microsoft cabinet format files. | More Info... |

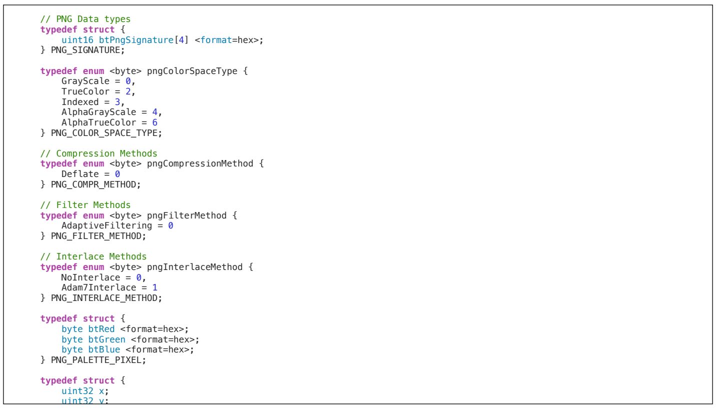
However, we have found a repository of input specifications. These stem from the 010 Editor, which uses so-called binary templates for structured display of complex inputs.



## Existing Input Format Specifications

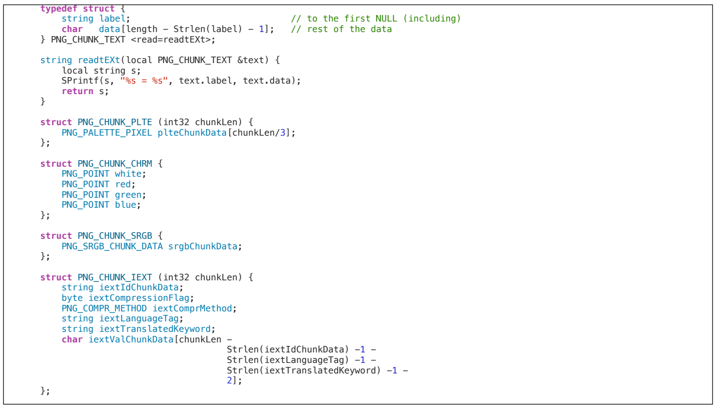
```
//------------------------------------------------
//--- 010 Editor Binary Template
//
//      File: PNG.bt
//   Authors: Kevin O. Grover, RCS, Mister Wu
//   Version: 2.3
//   Purpose: Parse PNG (Portable Network Graphics) and APNG (Animated Portable Network Graphics) image files.
//  Category: Image
// File Mask: *.png,*.apng
//  ID Bytes: 89 50 4E 47 //%PNG
//   History:
//   2.3   2018-08-17 K. Grover: Uniform type names.  Formatting/comments.
//   2.2   2017-10-20 Mister Wu: Initial support of APNG chunks: acTL, fcTL, fdAT.
//   2.1   2017-08-31 K. Grover: Better colors.  Alternate chunk colors.  Cleaned up messages.
//   2.0   2016-02-10 SweetScape Software: Merged in extra chunks from PNG12Template.bt, updated header for
repository submission.
//   1.1   2009-02-23 K. Grover: Decode IHDR and tEXt chunks.
//   1.0.1 2005-06-29 K. Grover: Fixed typos in comments.
//   1.0   2005-05-11 K. Grover: Initial version.
//
// This template was written to the PNG 1.2 Specification:
//     http://www.libpng.org/pub/png/spec/1.2/
//
// It includes chunks described in the APNG 1.0 Specification:
//     https://wiki.mozilla.org/APNG_Specification
//
// Note however, that it does not check nor parse all chunk subdata, so it
// should work with all future PNG specifications.
//
// Possible caveat: PNG encourages the chunk type to be mapped to
// strings of the form "[a-zA-Z]{4}".  However, it's not a requirement.
```

Here's an example – the PNG spec.

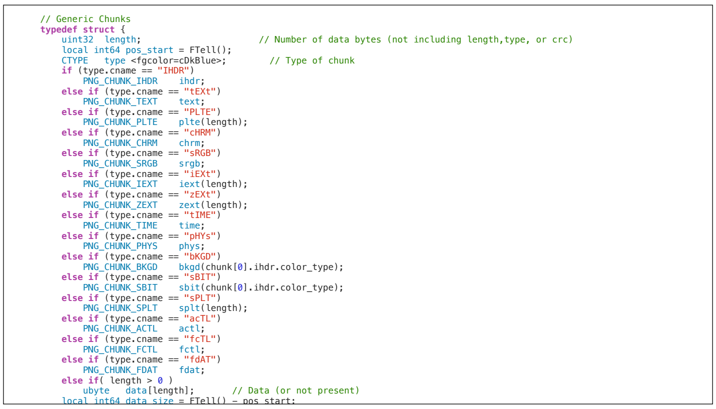


```
// PNG Data types
typedef struct {
    uint16 btPngSignature[4] <format=hex>;
} PNG_SIGNATURE;

typedef enum <byte> pngColorSpaceType {
    GrayScale = 0,
    TrueColor = 2,
    Indexed = 3,
    AlphaGrayScale = 4,
    AlphaTrueColor = 6
} PNG_COLOR_SPACE_TYPE;

// Compression Methods
typedef enum <byte> pngCompressionMethod {
    Deflate = 0
} PNG_COMPR_METHOD;

// Filter Methods
typedef enum <byte> pngFilterMethod {
    AdaptiveFiltering = 0
} PNG_FILTER_METHOD;

// Interlace Methods
typedef enum <byte> pngInterlaceMethod {
    NoInterlace = 0,
    Adam7Interlace = 1
} PNG_INTERLACE_METHOD;

typedef struct {
    byte btRed <format=hex>;
    byte btGreen <format=hex>;
    byte btBlue <format=hex>;
} PNG_PALETTE_PIXEL;

typedef struct {
    uint32 x;
    uint32 y;
```

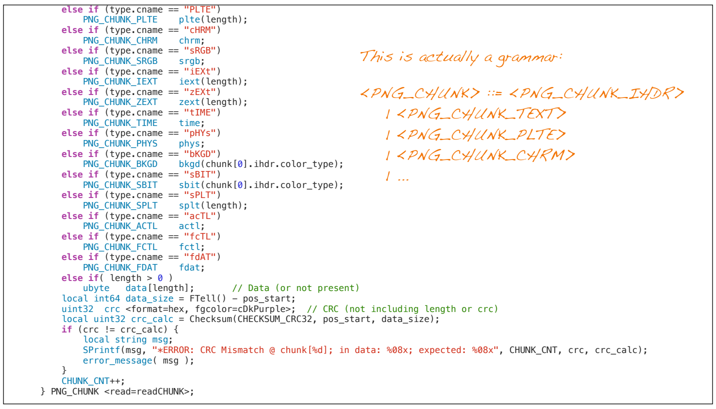
You see the individual elements (in a C-like syntax)



```
typedef struct {
    string label;                           // to the first NULL (including)
    char   data[length - Strlen(label) - 1];   // rest of the data
} PNG_CHUNK_TEXT <read=readtEXt>;

string readtEXt(local PNG_CHUNK_TEXT &text) {
    local string s;
    SPrintf(s, "%s = %s", text.label, text.data);
    return s;
}

struct PNG_CHUNK_PLTE (int32 chunkLen) {
    PNG_PALETTE_PIXEL plteChunkData[chunkLen/3];
};

struct PNG_CHUNK_CHRM {
    PNG_POINT white;
    PNG_POINT red;
    PNG_POINT green;
    PNG_POINT blue;
};

struct PNG_CHUNK_SRGB {
    PNG_SRGB_CHUNK_DATA srgbChunkData;
};

struct PNG_CHUNK_IEXT (int32 chunkLen) {
    string iextIdChunkData;
    byte iextCompressionFlag;
    PNG_COMPR_METHOD iextComprMethod;
    string iextLanguageTag;
    string iextTranslatedKeyword;
    char iextValChunkData[chunkLen -
                                   Strlen(iextIdChunkData) -1 -
                                   Strlen(iextLanguageTag) -1 -
                                   Strlen(iextTranslatedKeyword) -1 -
                                   2];
};
```



```
// Generic Chunks
typedef struct {
    uint32  length;                      // Number of data bytes (not including length,type, or crc)
    local int64 pos_start = FTell();
    CTYPE   type <fgcolor=cDkBlue>;        // Type of chunk
    if (type.cname == "IHDR")
        PNG_CHUNK_IHDR    ihdr;
    else if (type.cname == "tEXt")
        PNG_CHUNK_TEXT    text;
    else if (type.cname == "PLTE")
        PNG_CHUNK_PLTE    plte(length);
    else if (type.cname == "cHRM")
        PNG_CHUNK_CHRM    chrm;
    else if (type.cname == "sRGB")
        PNG_CHUNK_SRGB    srgb;
    else if (type.cname == "iEXt")
        PNG_CHUNK_IEXT    iext(length);
    else if (type.cname == "zEXt")
        PNG_CHUNK_ZEXT    zext(length);
    else if (type.cname == "tIME")
        PNG_CHUNK_TIME    time;
    else if (type.cname == "pHYs")
        PNG_CHUNK_PHYS    phys;
    else if (type.cname == "bKGD")
        PNG_CHUNK_BKGD    bkgd(chunk[0].ihdr.color_type);
    else if (type.cname == "sBIT")
        PNG_CHUNK_SBIT    sbit(chunk[0].ihdr.color_type);
    else if (type.cname == "sPLT")
        PNG_CHUNK_SPLT    splt(length);
    else if (type.cname == "acTL")
        PNG_CHUNK_ACTL    actl;
    else if (type.cname == "fcTL")
        PNG_CHUNK_FCTL    fctl;
    else if (type.cname == "fdAT")
        PNG_CHUNK_FDAT    fdat;
    else if( length > 0 )
        ubyte   data[length];       // Data (or not present)
    local int64 data_size = FTell() - pos_start;
```

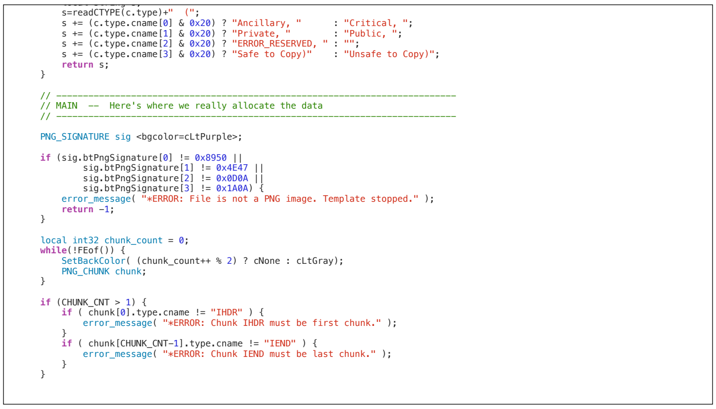


```
        else if (type.cname == "PLTE")
            PNG_CHUNK_PLTE    plte(length);
        else if (type.cname == "cHRM")
            PNG_CHUNK_CHRM    chrm;
        else if (type.cname == "sRGB")
            PNG_CHUNK_SRGB    srgb;
        else if (type.cname == "iEXt")
            PNG_CHUNK_IEXT    iext(length);
        else if (type.cname == "zEXt")
            PNG_CHUNK_ZEXT    zext(length);
        else if (type.cname == "tIME")
            PNG_CHUNK_TIME    time;
        else if (type.cname == "pHYs")
            PNG_CHUNK_PHYS    phys;
        else if (type.cname == "bKGD")
            PNG_CHUNK_BKGD    bkgd(chunk[0].ihdr.color_type);
        else if (type.cname == "sBIT")
            PNG_CHUNK_SBIT    sbit(chunk[0].ihdr.color_type);
        else if (type.cname == "sPLT")
            PNG_CHUNK_SPLT    splt(length);
        else if (type.cname == "acTL")
            PNG_CHUNK_ACTL    actl;
        else if (type.cname == "fcTL")
            PNG_CHUNK_FCTL    fctl;
        else if (type.cname == "fdAT")
            PNG_CHUNK_FDAT    fdat;
        else if( length > 0 )
            ubyte   data[length];       // Data (or not present)
        local int64 data_size = FTell() - pos_start;
        uint32  crc <format=hex, fgcolor=cDkPurple>;  // CRC (not including length or crc)
        local uint32 crc_calc = Checksum(CHECKSUM_CRC32, pos_start, data_size);
        if (crc != crc_calc) {
            local string msg;
            SPrintf(msg, "*ERROR: CRC Mismatch @ chunk[%d]; in data: %08x; expected: %08x", CHUNK_CNT, crc, crc_calc);
            error_message( msg );
        }
        CHUNK_CNT++;
    } PNG_CHUNK <read=readCHUNK>;
```

*This is actually a grammar:*

```
<PNG_CHUNK> ::= <PNG_CHUNK_IHDR>
    | <PNG_CHUNK_TEXT>
    | <PNG_CHUNK_PLTE>
    | <PNG_CHUNK_CHRM>
    | ...
```

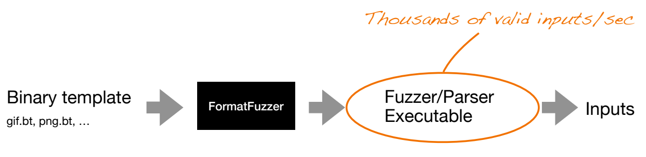
Technically, this is a grammar – but it doesn't look like it. Plus, there's extra code.



```
        s=readCTYPE(c.type)+"  (";
        s += (c.type.cname[0] & 0x20) ? "Ancillary, "      : "Critical, ";
        s += (c.type.cname[1] & 0x20) ? "Private, "        : "Public, ";
        s += (c.type.cname[2] & 0x20) ? "ERROR_RESERVED, " : "";
        s += (c.type.cname[3] & 0x20) ? "Safe to Copy)"    : "Unsafe to Copy)";
        return s;
    }

    // ---------------------------------------------------------------------------
    // MAIN  --  Here's where we really allocate the data
    // ---------------------------------------------------------------------------

    PNG_SIGNATURE sig <bgcolor=cLtPurple>;

    if (sig.btPngSignature[0] != 0x8950 ||
            sig.btPngSignature[1] != 0x4E47 ||
            sig.btPngSignature[2] != 0x0D0A ||
            sig.btPngSignature[3] != 0x1A0A) {
        error_message( "*ERROR: File is not a PNG image. Template stopped." );
        return -1;
    }

    local int32 chunk_count = 0;
    while(!FEof()) {
        SetBackColor( (chunk_count++ % 2) ? cNone : cLtGray);
        PNG_CHUNK chunk;
    }

    if (CHUNK_CNT > 1) {
        if ( chunk[0].type.cname != "IHDR" ) {
            error_message( "*ERROR: Chunk IHDR must be first chunk." );
        }
        if ( chunk[CHUNK_CNT-1].type.cname != "IEND" ) {
            error_message( "*ERROR: Chunk IEND must be last chunk." );
        }
    }
```

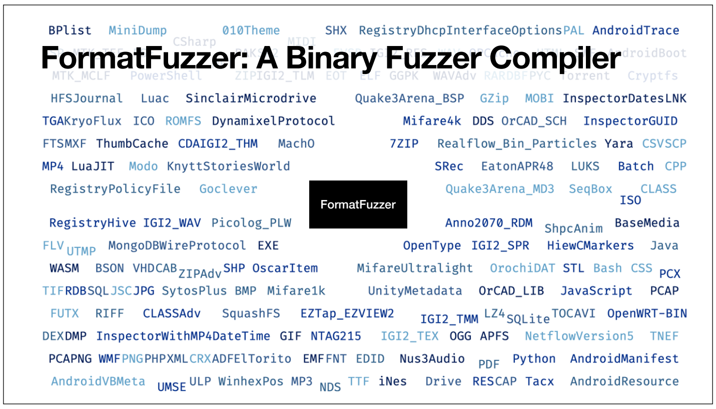
This extra code checks for semantic constraints that couldn't be expressed in a grammar.

## FormatFuzzer: A Binary Fuzzer Compiler



*Thousands of valid inputs/sec*

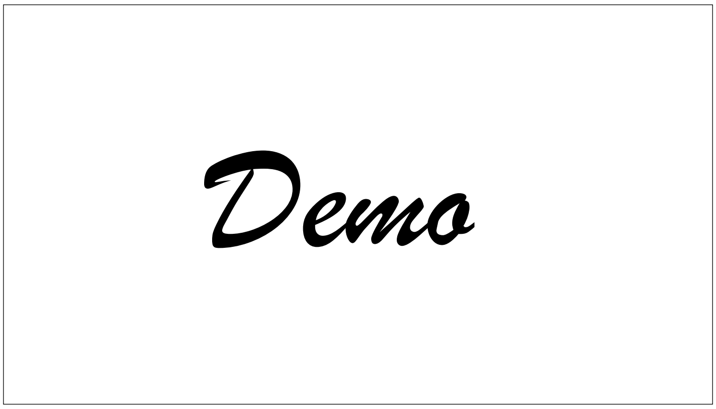
We have built a fuzzer that takes such a template and compiles it into a fuzzer.



## FormatFuzzer: A Binary Fuzzer Compiler

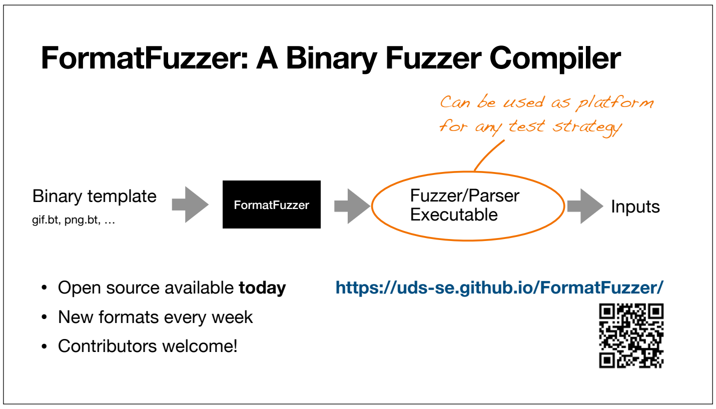
BPlist MiniDump 010Theme SHX RegistryDhcpInterfaceOptions PAL AndroidTrace
CSharp MIDI AndroidBoot
MTK_MCLF PowerShell ZIPIGI2_TLM EOT ELF GGPK WAVAdv RARDBF PYC Torrent Cryptfs
HFSJournal Luac SinclairMicrodrive Quake3Arena_BSP GZip MOBI InspectorDatesLNK
TGAKryoFlux ICO ROMFS DynamixelProtocol Mifare4k DDS OrCAD_SCH InspectorGUID
FTSMXF ThumbCache CDAIGI2_THM MachO 7ZIP Realflow_Bin_Particles Yara CSVSCP
MP4 LuaJIT Modo KnyttStoriesWorld SRec EatonAPR48 LUKS Batch CPP
RegistryPolicyFile Goclever FormatFuzzer Quake3Arena_MD3 SeqBox CLASS ISO
RegistryHive IGI2_WAV Picolog_PLW Anno2070_RDM ShpcAnim BaseMedia
FLV UTMP MongoDBWireProtocol EXE OpenType IGI2_SPR HiewCMarkers Java
WASM BSON VHDCAB ZIPAdv SHP OscarItem MifareUltralight OrochiDAT STL Bash CSS PCX
TIF RDB SQL JSC JPG SytosPlus BMP Mifare1k UnityMetadata OrCAD_LIB JavaScript PCAP
FUTX RIFF CLASSAdv SquashFS EZTap_EZVIEW2 IGI2_TMM LZ4 SQLite TOCAVI OpenWRT-BIN
DEX DMP InspectorWithMP4DateTime GIF NTAG215 IGI2_TEX OGG APFS NetflowVersion5 TNEF
PCAPNG WMF PNG PHP XML CRX ADF ElTorito EMF FNT EDID Nus3Audio PDF Python AndroidManifest
AndroidVBMeta UMSE ULP WinhexPos MP3 NDS TTF iNes Drive RESCAP Tacx AndroidResource

The plan is that you can go and use it for all the 170 formats around (and many more)



Demo

So let me show it to you :-)

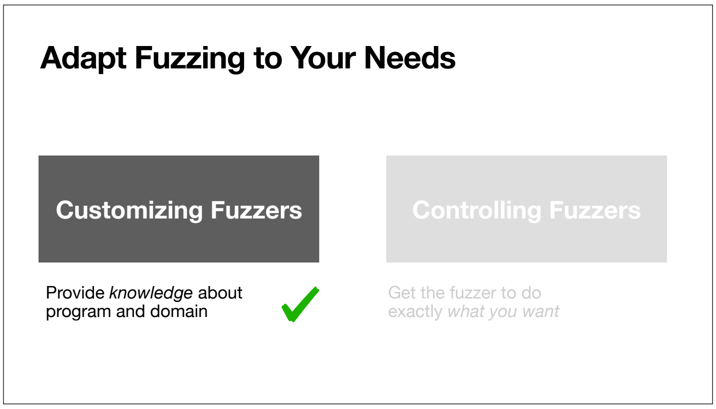


## FormatFuzzer: A Binary Fuzzer Compiler

- Open source available **today**
- New formats every week
- Contributors welcome!

**https://uds-se.github.io/FormatFuzzer/**

Good news: We have opened up the repo today, and you can try it out yourself. It will still take a couple of weeks until this is ready for prime time, but feel free to peek into it now. Contributors welcome!



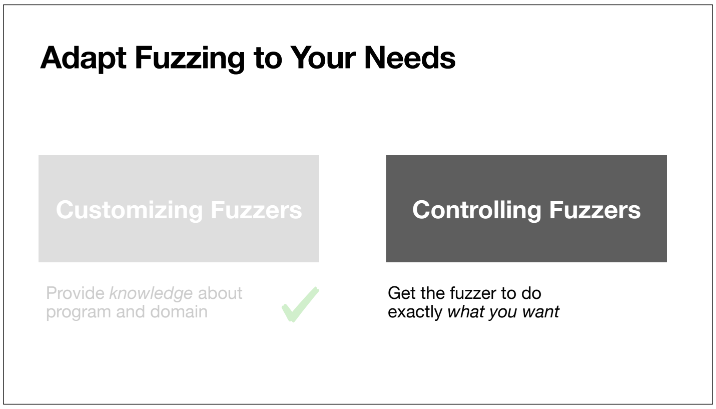
## Adapt Fuzzing to Your Needs

**Customizing Fuzzers**

Provide *knowledge* about program and domain ✓

**Controlling Fuzzers**

Get the fuzzer to do exactly *what you want*



## Adapt Fuzzing to Your Needs

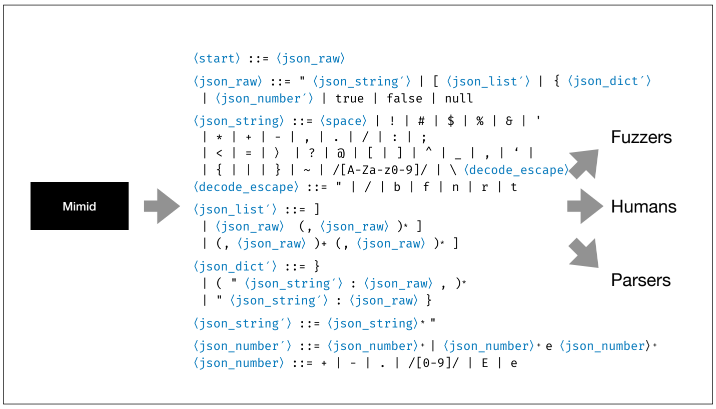
**Customizing Fuzzers**

Provide *knowledge* about program and domain ✓

**Controlling Fuzzers**

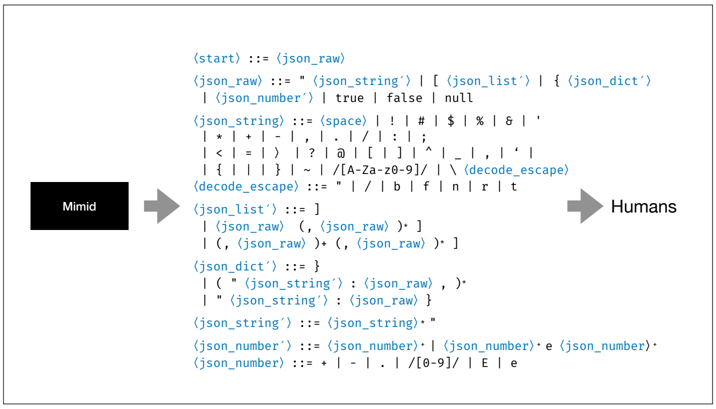
Get the fuzzer to do exactly *what you want*

Now a bit about controlling fuzzers



```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | [ ⟨json_list′⟩ | { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | ⟩  | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* "
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

Something we rarely talk about is that grammars give you lots of control over what should be created.

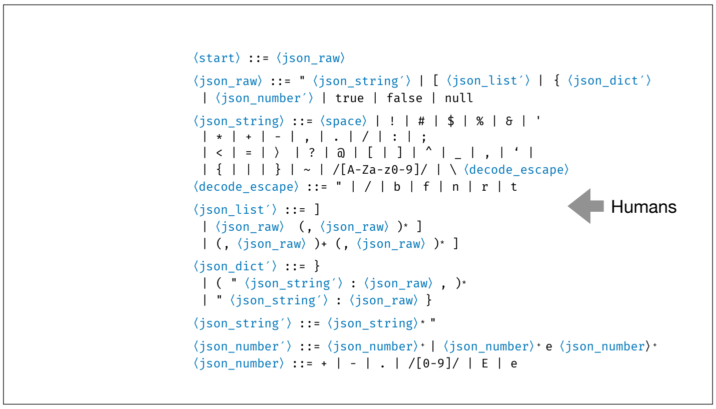


Mimid

```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | [ ⟨json_list′⟩ | { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | ⟩  | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* "
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

Humans

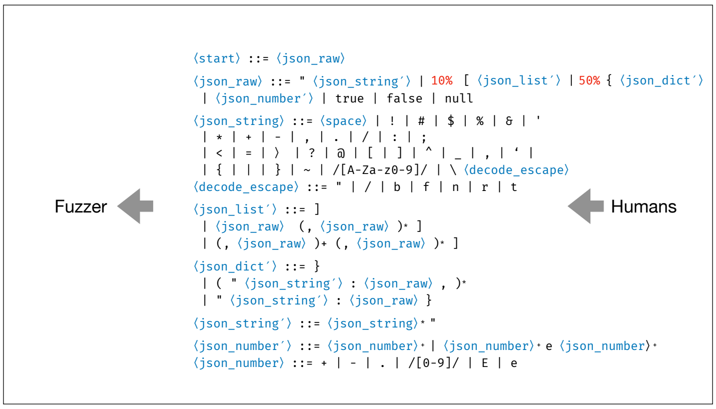
The extracted grammars are well structured and human readable as you can see in this grammar extracted from a JSON parser.



```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | [ ⟨json_list′⟩ | { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | ⟩  | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* "
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

Humans

Humans can **edit** these grammars.

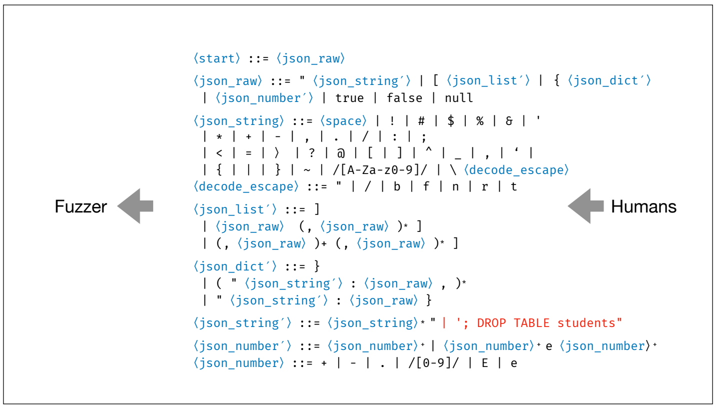


```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | 10%  [ ⟨json_list′⟩ | 50% { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | ⟩  | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* "
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

Fuzzer

Humans

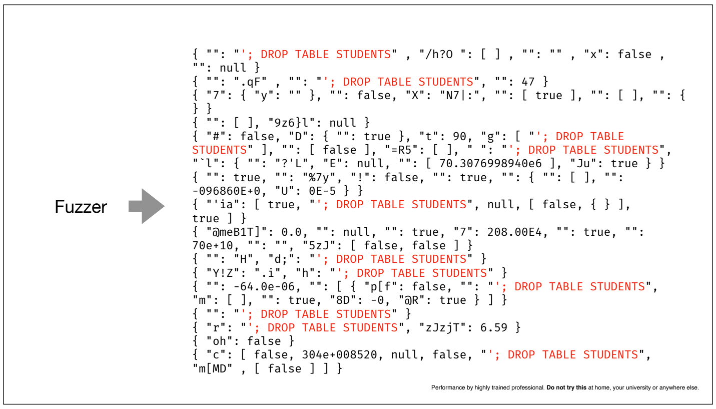
For instance, by assigning probabilities to individual productions.



Fuzzer
Humans

```
⟨start⟩ ::= ⟨json_raw⟩
⟨json_raw⟩ ::= " ⟨json_string′⟩ | [ ⟨json_list′⟩ | { ⟨json_dict′⟩
 | ⟨json_number′⟩ | true | false | null
⟨json_string⟩ ::= ⟨space⟩ | ! | # | $ | % | & | '
 | * | + | - | , | . | / | : | ;
 | < | = | ⟩  | ? | @ | [ | ] | ^ | _ | , | ‘ |
 | { | | | } | ~ | /[A-Za-z0-9]/ | \ ⟨decode_escape⟩
⟨decode_escape⟩ ::= " | / | b | f | n | r | t
⟨json_list′⟩ ::= ]
 | ⟨json_raw⟩  (, ⟨json_raw⟩ )* ]
 | (, ⟨json_raw⟩ )+ (, ⟨json_raw⟩ )* ]
⟨json_dict′⟩ ::= }
 | ( " ⟨json_string′⟩ : ⟨json_raw⟩ , )*
 | " ⟨json_string′⟩ : ⟨json_raw⟩ }
⟨json_string′⟩ ::= ⟨json_string⟩* " | '; DROP TABLE students"
⟨json_number′⟩ ::= ⟨json_number⟩+ | ⟨json_number⟩+ e ⟨json_number⟩+
⟨json_number⟩ ::= + | - | . | /[0-9]/ | E | e
```

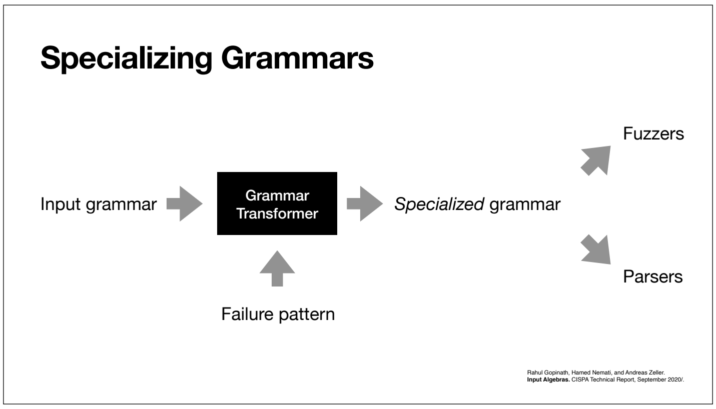
Or by inserting magic strings that program analysis would have a hard time finding out.



```
{ "": "'; DROP TABLE STUDENTS" , "/h?O ": [ ] , "": "" , "x": false ,
"": null }
{ "": ".qF" , "": "'; DROP TABLE STUDENTS", "": 47 }
{ "7": { "y": "" }, "": false, "X": "N7|:", "": [ true ], "": [ ], "": {
} }
{ "": [ ], "9z6}l": null }
{ "#": false, "D": { "": true }, "t": 90, "g": [ "'; DROP TABLE
STUDENTS" ], "": [ false ], "=R5": [ ], " ": "'; DROP TABLE STUDENTS",
"`l": { "": "?'L", "E": null, "": [ 70.3076998940e6 ], "Ju": true } }
{ "": true, "": "%7y", "!": false, "": true, "": { "": [ ], "":
-096860E+0, "U": 0E-5 } }
{ "'ia": [ true, "'; DROP TABLE STUDENTS", null, [ false, { } ],
true ] }
{ "@meB1T]": 0.0, "": null, "": true, "7": 208.00E4, "": true, "":
70e+10, "": "", "5zJ": [ false, false ] }
{ "": "H", "d;": "'; DROP TABLE STUDENTS" }
{ "Y!Z": ".i", "h": "'; DROP TABLE STUDENTS" }
{ "": -64.0e-06, "": [ { "p[f": false, "": "'; DROP TABLE STUDENTS",
"m": [ ], "": true, "8D": -0, "@R": true } ] }
{ "": "'; DROP TABLE STUDENTS" }
{ "r": "'; DROP TABLE STUDENTS", "zJzjT": 6.59 }
{ "oh": false }
{ "c": [ false, 304e+008520, null, false, "'; DROP TABLE STUDENTS",
"m[MD" , [ false ] ] }
```

Performance by highly trained professional. **Do not try this** at home, your university or anywhere else.

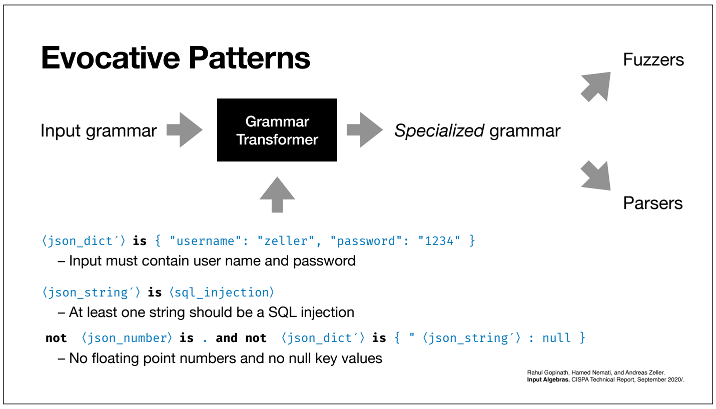
This change to the grammar injects SQL statements everywhere. Do not do this at home, folks – thank you.



## Specializing Grammars

Rahul Gopinath, Hamed Nemati, and Andreas Zeller.
**Input Algebras.** CISPA Technical Report, September 2020/.

In our most recent work, we have introduced **grammar transformers** that take a grammar and specialize it towards a specific goal.



## Evocative Patterns

Input grammar → Grammar Transformer → *Specialized* grammar → Fuzzers / Parsers

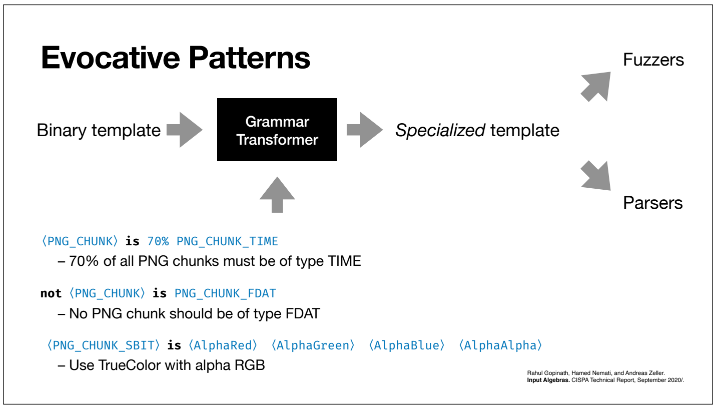
```
⟨json_dict′⟩ is { "username": "zeller", "password": "1234" }
```
– Input must contain user name and password

```
⟨json_string′⟩ is ⟨sql_injection⟩
```
– At least one string should be a SQL injection

```
not  ⟨json_number⟩ is . and not  ⟨json_dict′⟩ is { " ⟨json_string′⟩ : null }
```
– No floating point numbers and no null key values

Rahul Gopinath, Hamed Nemati, and Andreas Zeller.
**Input Algebras.** CISPA Technical Report, September 2020/.

Using special expressions, we can control what should be produced. The result is another grammar, which can be used with any grammar-based fuzzer.



## Evocative Patterns

Binary template → Grammar Transformer → *Specialized* template → Fuzzers / Parsers

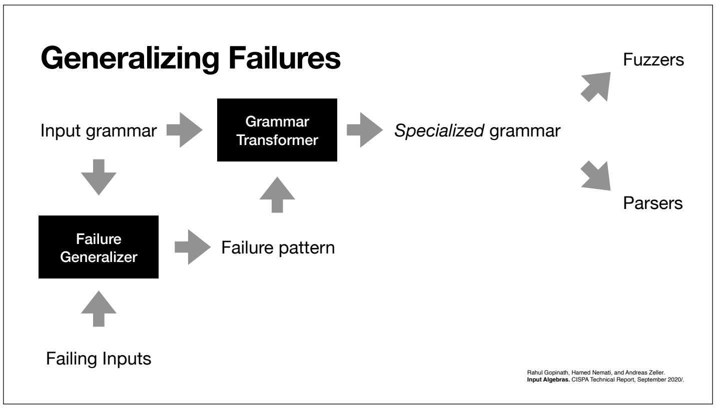
```
⟨PNG_CHUNK⟩ is 70% PNG_CHUNK_TIME
```
– 70% of all PNG chunks must be of type TIME

```
not ⟨PNG_CHUNK⟩ is PNG_CHUNK_FDAT
```
– No PNG chunk should be of type FDAT

```
⟨PNG_CHUNK_SBIT⟩ is ⟨AlphaRed⟩  ⟨AlphaGreen⟩  ⟨AlphaBlue⟩  ⟨AlphaAlpha⟩
```
– Use TrueColor with alpha RGB

Rahul Gopinath, Hamed Nemati, and Andreas Zeller.
**Input Algebras.** CISPA Technical Report, September 2020/.

This includes our own FormatFuzzer, by the way; so we're working hard on also controlling FormatFuzzer.



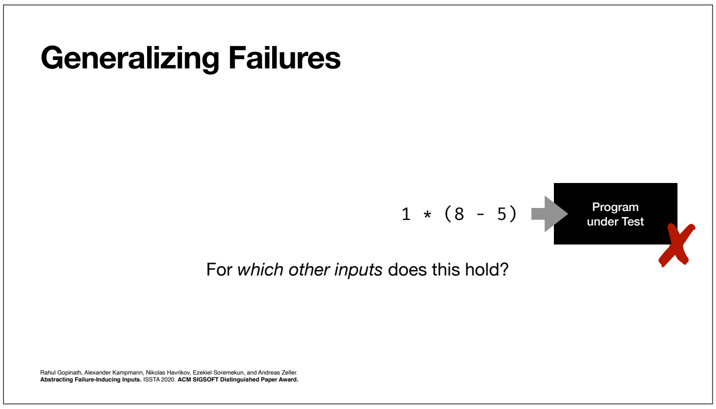
## Generalizing Failures

Input grammar → Grammar Transformer → *Specialized* grammar → Fuzzers / Parsers

Failing Inputs → Failure Generalizer → Failure pattern

Rahul Gopinath, Hamed Nemati, and Andreas Zeller.
**Input Algebras.** CISPA Technical Report, September 2020/.

Interestingly, these patterns we use to control the fuzzer can actually come from **earlier failures**

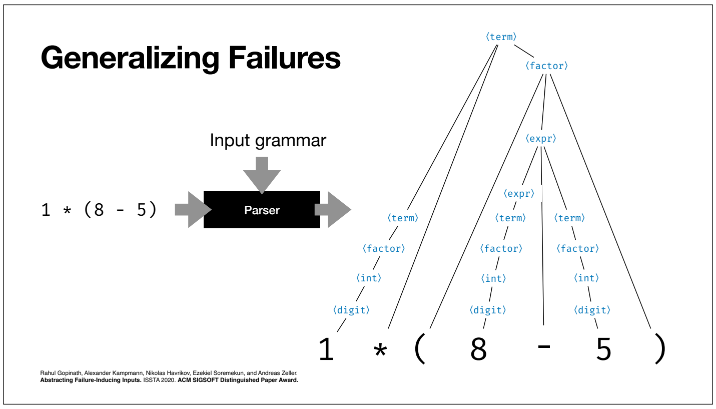


## Generalizing Failures

For *which other inputs* does this hold?

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller. **Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

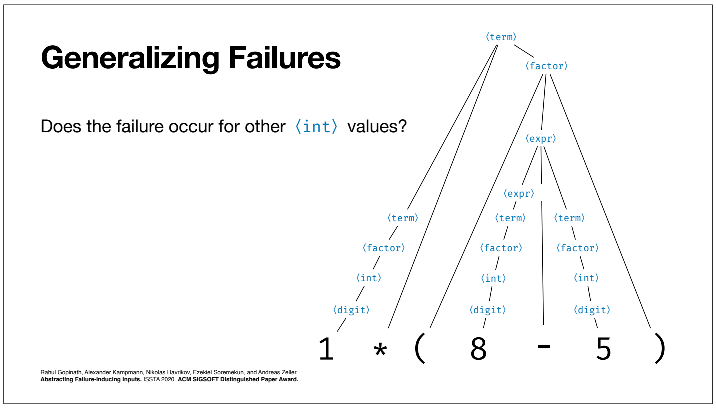
When you do fuzzing, you'll find single inputs that cause failures. But are these the only inputs?



## Generalizing Failures

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller. **Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

We want to know the **set of inputs** that causes the failure – in other words, the language. To this end, we parse the input into a tree.

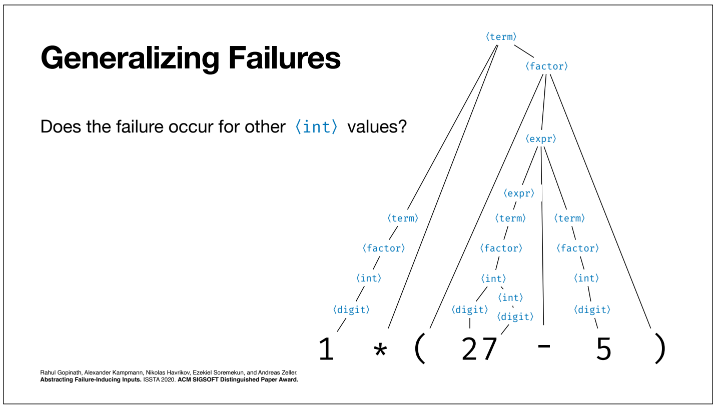


## Generalizing Failures

Does the failure occur for other ⟨int⟩ values?

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller. **Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

To find out whether the failure occurs for other integer values too, …

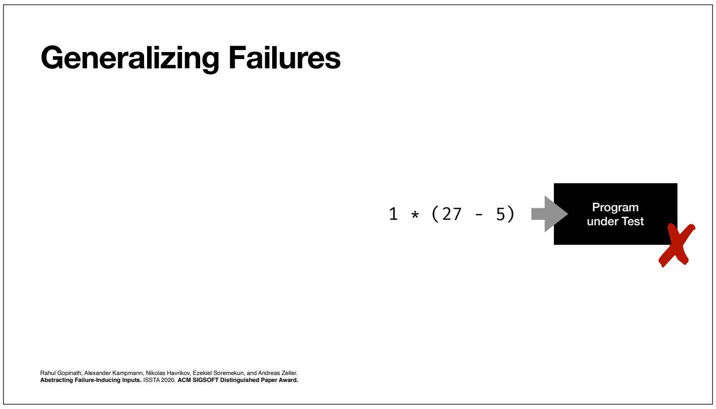


## Generalizing Failures

Does the failure occur for other ⟨int⟩ values?

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller.
**Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

… we replace parts of the parse tree (8) by newly generated alternatives (27).

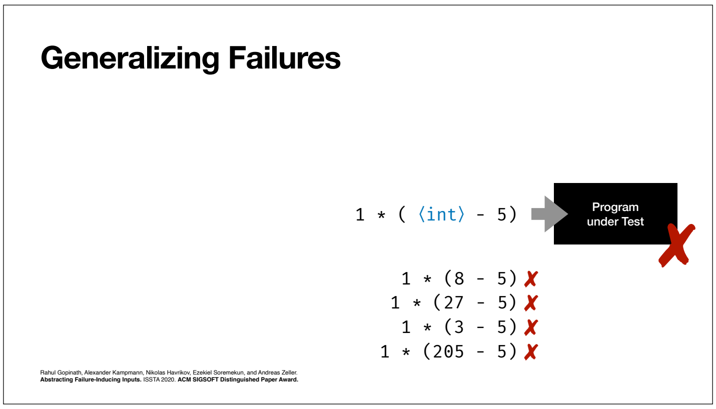


## Generalizing Failures

```
1 * (27 - 5)
```

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller.
**Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

and find that this one fails as well.

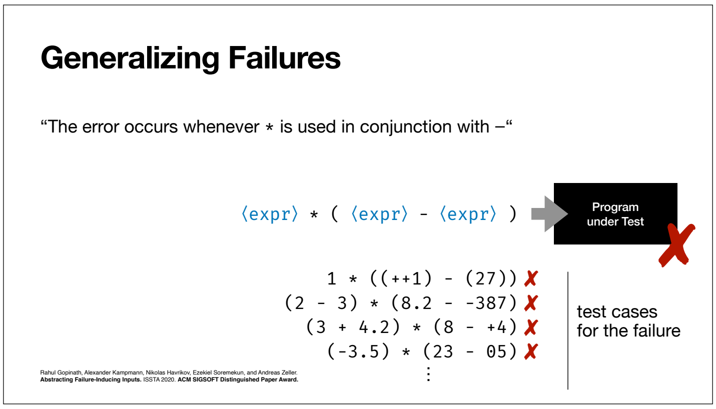


## Generalizing Failures

```
1 * ( ⟨int⟩ - 5)

   1 * (8 - 5) ✗
  1 * (27 - 5) ✗
   1 * (3 - 5) ✗
 1 * (205 - 5) ✗
```

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller.
**Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

Reliably reachly, the program fails for any integer in this position. So we can come up with an abstract pattern that represents the set of failing inputs.



## Generalizing Failures

“The error occurs whenever * is used in conjunction with –“

```
⟨expr⟩ * ( ⟨expr⟩ - ⟨expr⟩ )
```

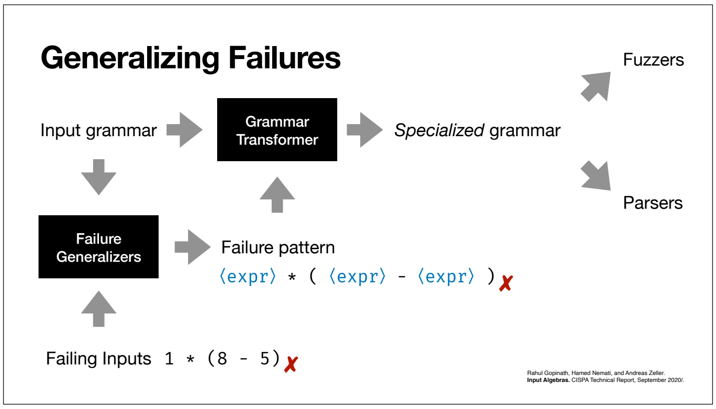
Program under Test ✗

```
   1 * ((++1) - (27)) ✗
(2 - 3) * (8.2 - -387) ✗
 (3 + 4.2) * (8 - +4) ✗
  (-3.5) * (23 - 05) ✗
          ⋮
```

test cases for the failure

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller. **Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

By repeating this, we can come up with a general pattern of which **all** instantiations cause the failure. These instantiations also serve as test cases for validating a fix.



## Generalizing Failures

Input grammar

Grammar Transformer

*Specialized* grammar

Fuzzers

Parsers

Failure Generalizers

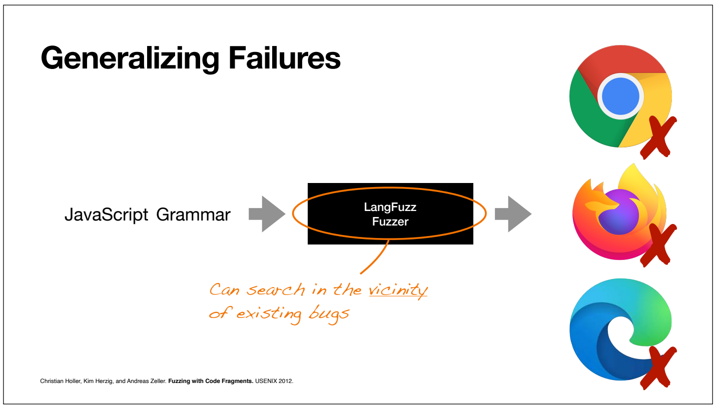
Failure pattern

```
⟨expr⟩ * ( ⟨expr⟩ - ⟨expr⟩ ) ✗
```

Failing Inputs `1 * (8 - 5)` ✗

Rahul Gopinath, Hamed Nemati, and Andreas Zeller. **Input Algebras.** CISPA Technical Report, September 2020/.

This means that you can start with a failing input, generalize this into a pattern, and then create a specialized grammar which includes this pattern again and again.



## Generalizing Failures

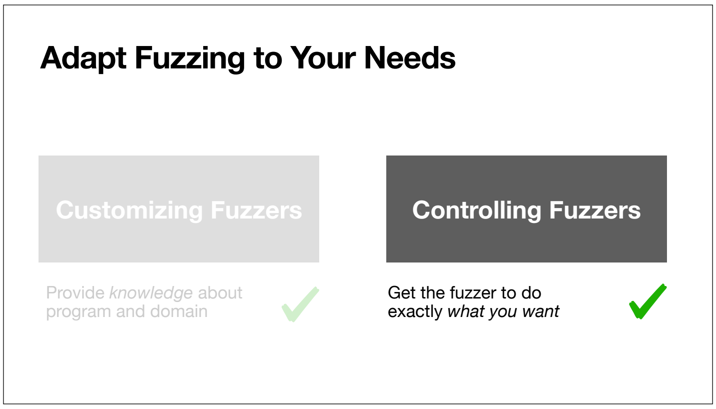
JavaScript Grammar

LangFuzz Fuzzer

Can search in the vicinity of existing bugs

Christian Holler, Kim Herzig, and Andreas Zeller. **Fuzzing with Code Fragments.** USENIX 2012.

For a fuzzer, this means that we can search in the vicinity of existing bugs – and this is tremendously successful.



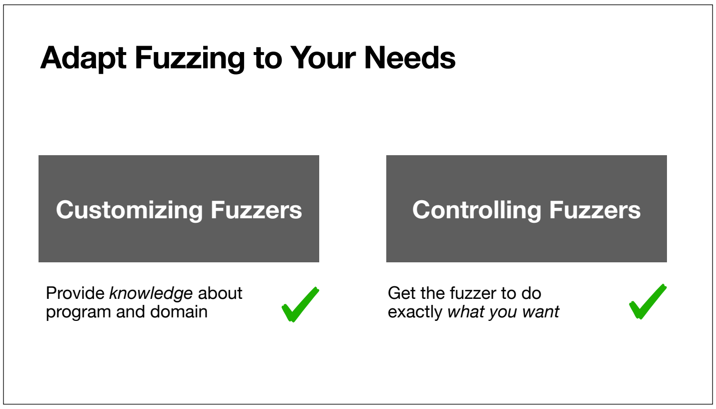
## Adapt Fuzzing to Your Needs

Customizing Fuzzers

Provide *knowledge* about program and domain ✓

Controlling Fuzzers

Get the fuzzer to do exactly *what you want* ✓



## Adapt Fuzzing to Your Needs

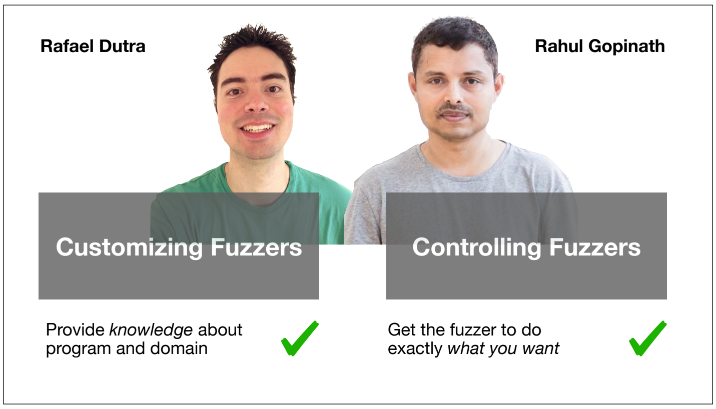
Customizing Fuzzers

Provide *knowledge* about program and domain ✓

Controlling Fuzzers

Get the fuzzer to do exactly *what you want* ✓

So this was a peek into what we do for taming fuzzers – and I think there's quite some perspective for fuzzing here.



Rafael Dutra

Rahul Gopinath

Customizing Fuzzers

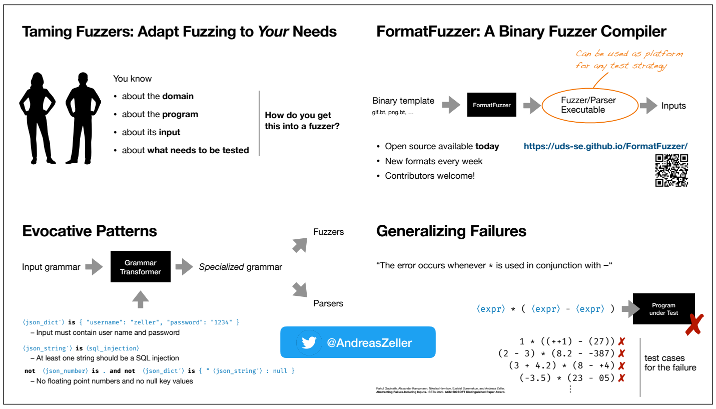
Provide *knowledge* about program and domain ✓

Controlling Fuzzers

Get the fuzzer to do exactly *what you want* ✓

Let me point out the masterminds behind these works. Rafael Dutra designed and write FormatFuzzer, Rahul Gopinath did the work on specializing grammars towards (generalized) patterns. They will be on the job market soon!

This are just two of the researchers at CISPA. If you don't know CISPA, this is a Helmholtz Center that is set to become Europe's largest center focusing on security fundamental research, with huge base funding and more than 800 positions for researchers at all levels. We are growing, we are hiring.



### Taming Fuzzers: Adapt Fuzzing to *Your* Needs

You know

- about the **domain**
- about the **program**
- about its **input**
- about **what needs to be tested**

**How do you get this into a fuzzer?**

### FormatFuzzer: A Binary Fuzzer Compiler

Binary template
gif.bt, png.bt, …
FormatFuzzer
Fuzzer/Parser Executable
Inputs
Can be used as platform for any test strategy

- Open source available **today**
- New formats every week
- Contributors welcome!

https://uds-se.github.io/FormatFuzzer/

### Evocative Patterns

Input grammar
Grammar Transformer
*Specialized* grammar
Fuzzers
Parsers

```
⟨json_dict′⟩ is { "username": "zeller", "password": "1234" }
```
– Input must contain user name and password

```
⟨json_string′⟩ is ⟨sql_injection⟩
```
– At least one string should be a SQL injection

```
not ⟨json_number⟩ is . and not ⟨json_dict′⟩ is { " ⟨json_string′⟩ : null }
```
– No floating point numbers and no null key values

@AndreasZeller

### Generalizing Failures

"The error occurs whenever * is used in conjunction with –"

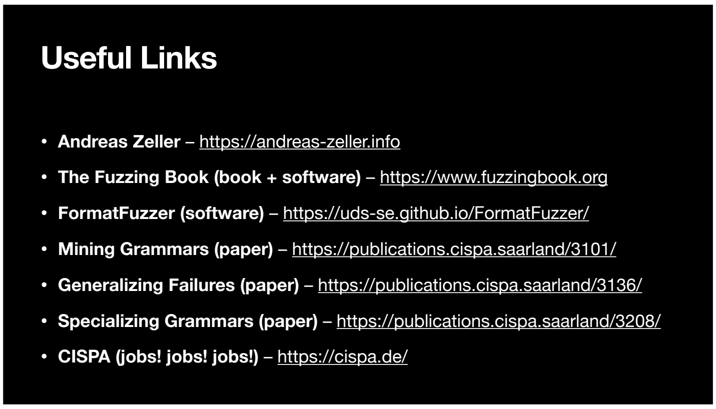
```
⟨expr⟩ * ( ⟨expr⟩ - ⟨expr⟩ )
```

```
   1 * ((++1) - (27)) ✗
(2 - 3) * (8.2 - -387) ✗
 (3 + 4.2) * (8 - +4) ✗
   (-3.5) * (23 - 05) ✗
          ⋮
```

test cases for the failure

Rahul Gopinath, Alexander Kampmann, Nikolas Havrikov, Ezekiel Soremekun, and Andreas Zeller. **Abstracting Failure-Inducing Inputs.** ISSTA 2020. **ACM SIGSOFT Distinguished Paper Award.**

That's all! If you like this work, and want to know more, follow me on Twitter or visit my homepage at https://andreas-zeller.info/. See you!



## Useful Links

- **Andreas Zeller** – https://andreas-zeller.info
- **The Fuzzing Book (book + software)** – https://www.fuzzingbook.org
- **FormatFuzzer (software)** – https://uds-se.github.io/FormatFuzzer/
- **Mining Grammars (paper)** – https://publications.cispa.saarland/3101/
- **Generalizing Failures (paper)** – https://publications.cispa.saarland/3136/
- **Specializing Grammars (paper)** – https://publications.cispa.saarland/3208/
- **CISPA (jobs! jobs! jobs!)** – https://cispa.de/