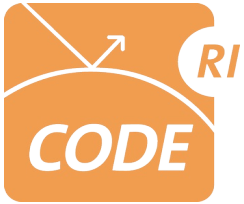
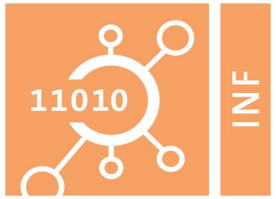
TEPHRA: Principled Discovery of Fuzzer Limitations

Vasil Sarafov, David Markvica, Stefan Brunthaler

ASE'25, Seoul, South Korea, November 16-20, 2025





Universität der Bundeswehr München
Fakultät für Informatik

Motivation

Fuzzers are valuable tools
because they find **real bugs**
in **real-world systems**.

Motivation

Bug finding is **undecidable**,
fuzzers rely on **heuristics**.

In Fact...

0) Design statement

American Fuzzy Lop does its best not to focus on any singular principle of operation and not be a proof-of-concept for any specific theory. The tool can be thought of as a **collection of hacks** that have been **tested in practice**, **found to be surprisingly effective**, and have been implemented in the simplest, most robust way I could think of at the time.

AFL, Michał Zalewski, 2013

Motivation

What are the limits of current (coverage-guided) fuzzing heuristics?

Motivation

How do we find those limits?

Motivation

Unlike formal methods, fuzz testing lacks a fundamental theory.

Motivation

Unlike formal methods, fuzz testing lacks a fundamental theory.

We rely entirely on empirical observations to evaluate its effectiveness.

Motivation

Existing empirical methods are based on (benchmark) program suites.

Semantic Space of all possible program behaviors

Motivation

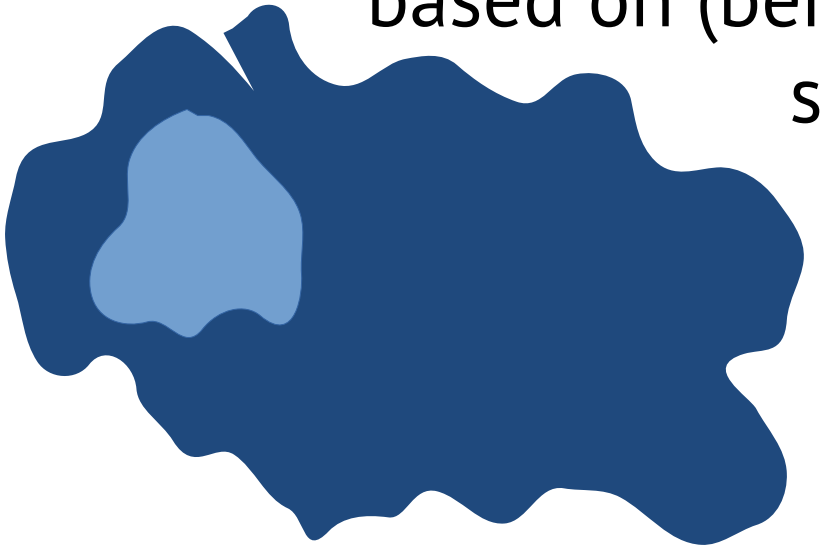
Existing empirical methods are based on (benchmark) program suites.

Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.

Motivation

Existing empirical methods are based on (benchmark) program suites.

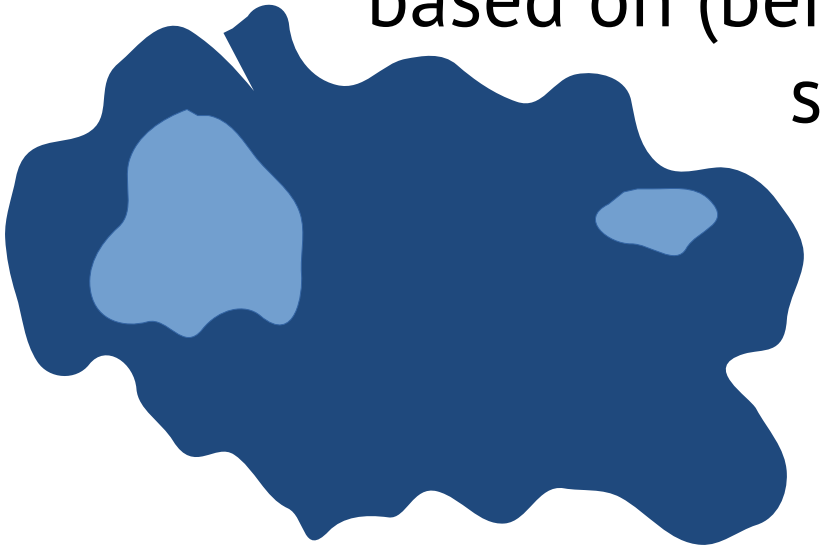


Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.

Motivation

Existing empirical methods are based on (benchmark) program suites.

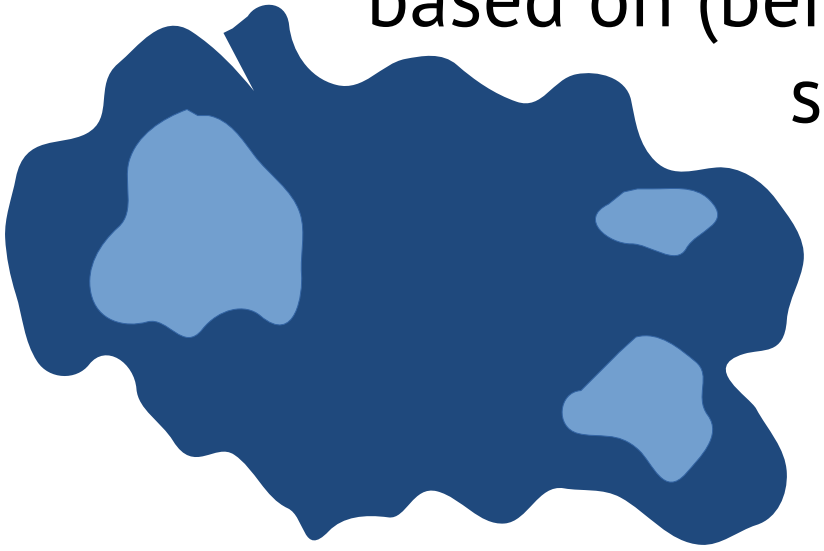


Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.

Motivation

Existing empirical methods are based on (benchmark) program suites.

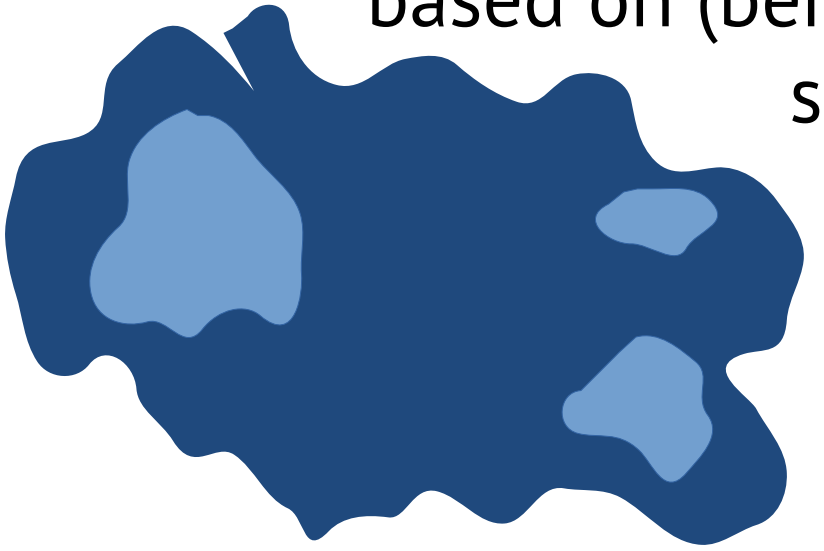


Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.

Motivation

Existing empirical methods are based on (benchmark) program suites.

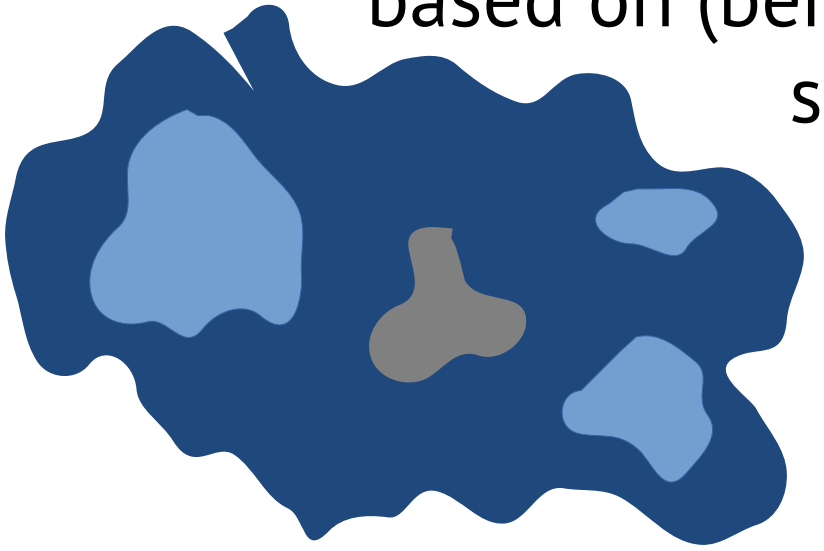


Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.
- Risk of overfitting/bias.

Motivation

Existing empirical methods are based on (benchmark) program suites.



Semantic Space of all possible program behaviors

- Piecewise static exploration of the semantic space.
- Risk of overfitting/bias.
- **What if my PUT is behaviorally very different?**

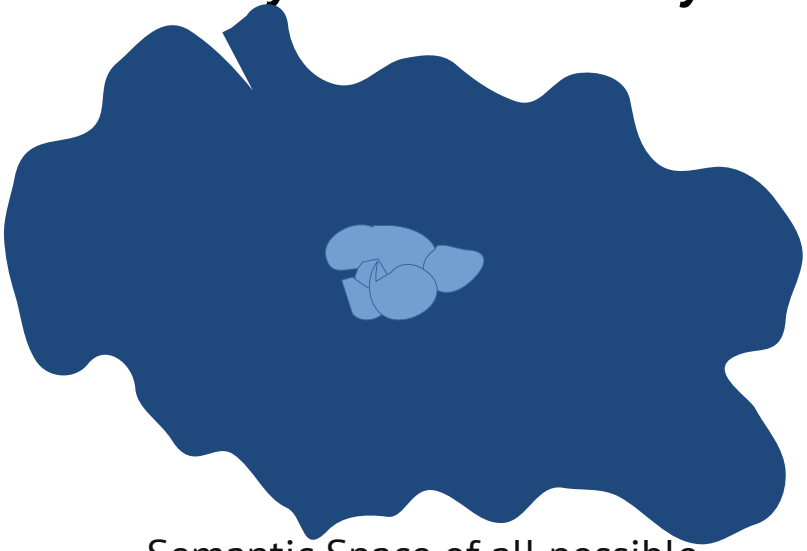
Motivation

What about exploring the semantic space more systematically?

Semantic Space of all possible program behaviors

Motivation

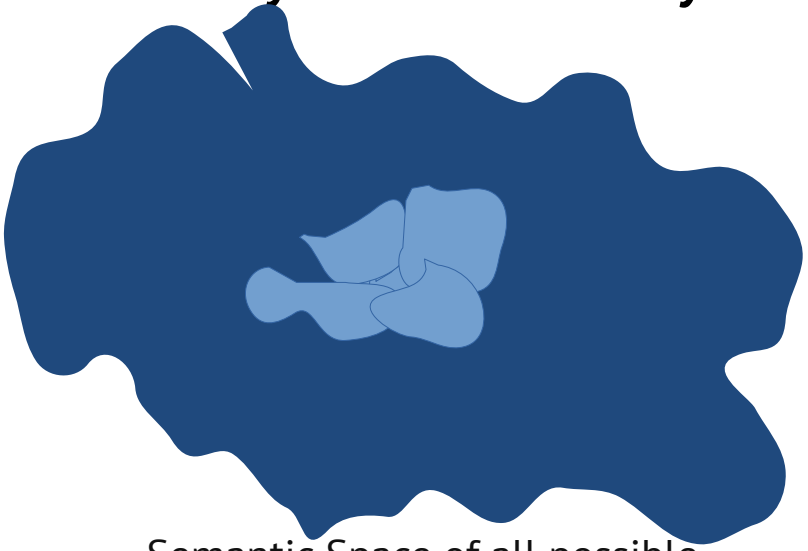
What about exploring the semantic space more systematically?



Semantic Space of all possible program behaviors

Motivation

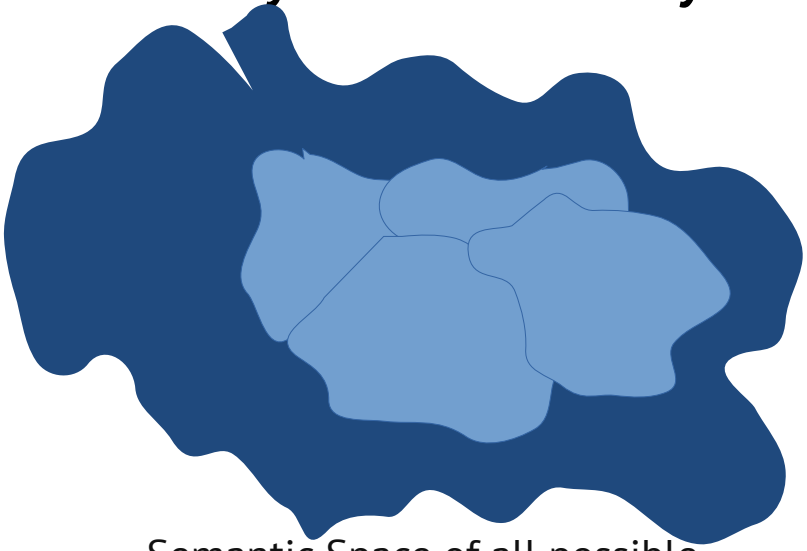
What about exploring the semantic space more systematically?



Semantic Space of all possible program behaviors

Motivation

What about exploring the semantic space more systematically?

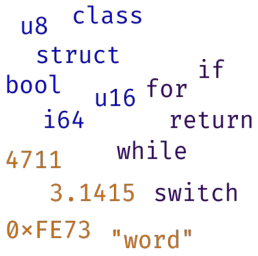


Semantic Space of all possible program behaviors

TEPHRA

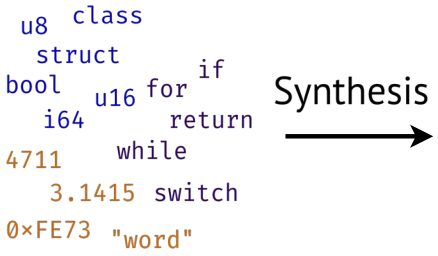
Principled Methodology to Empirically Discover Fuzzer Limitations

TEPHRA



Semantics

TEPHRA

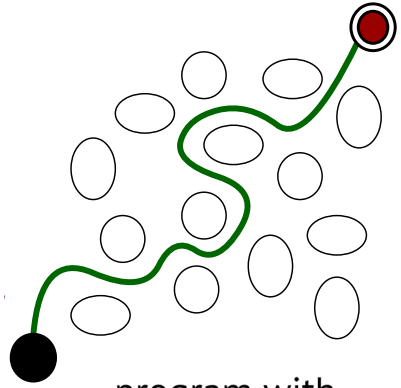

u8 class
struct
bool u16 for if
i64 return
4711 while
3.1415 switch
0×FE73 "word"
Synthesis
Semantics

TEPHRA

```
u8  class
  struct
bool     u16 for  if
   i64        return
4711     while
   3.1415  switch
0×FE73  "word"
```

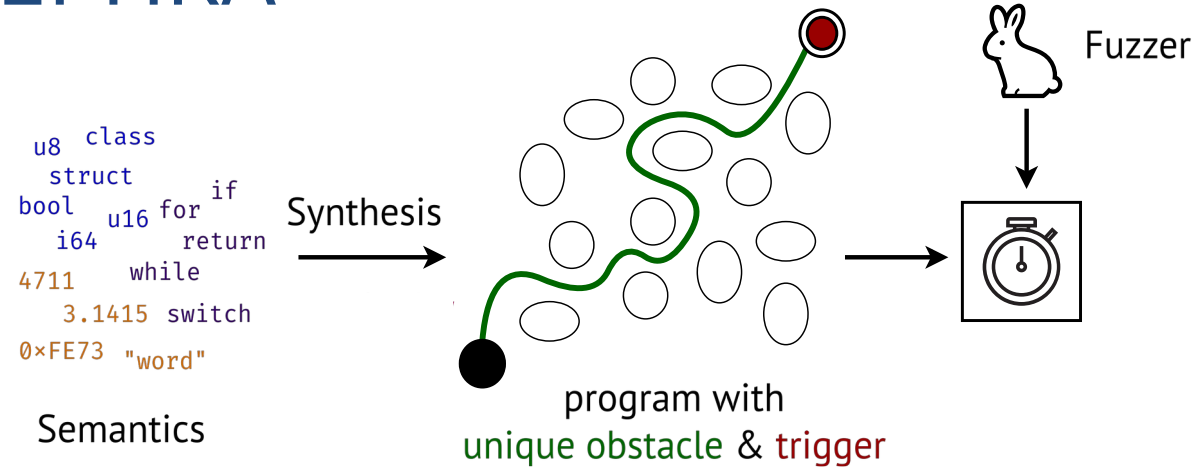
Semantics

Synthesis

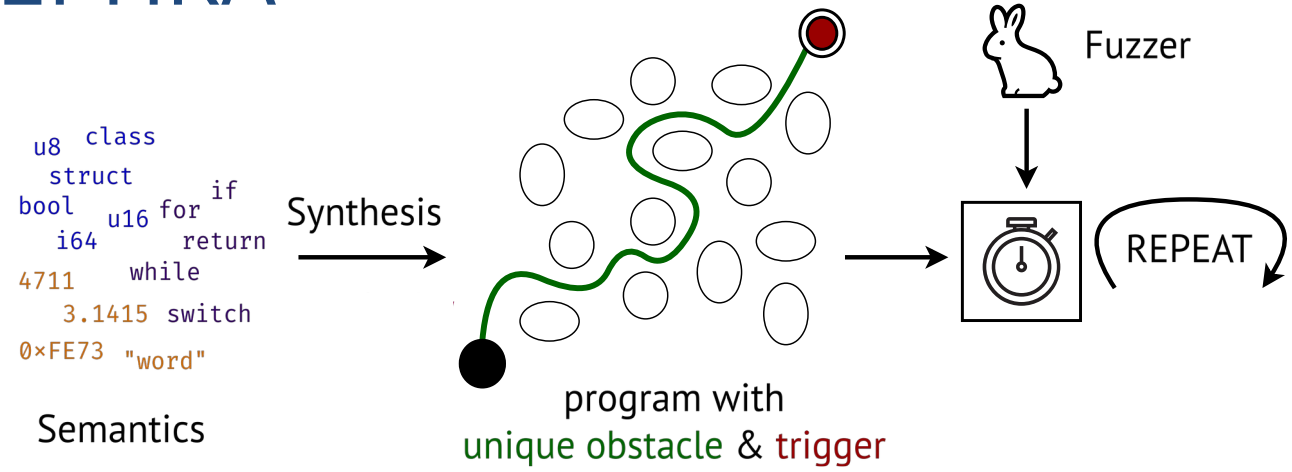


program with
unique obstacle & trigger

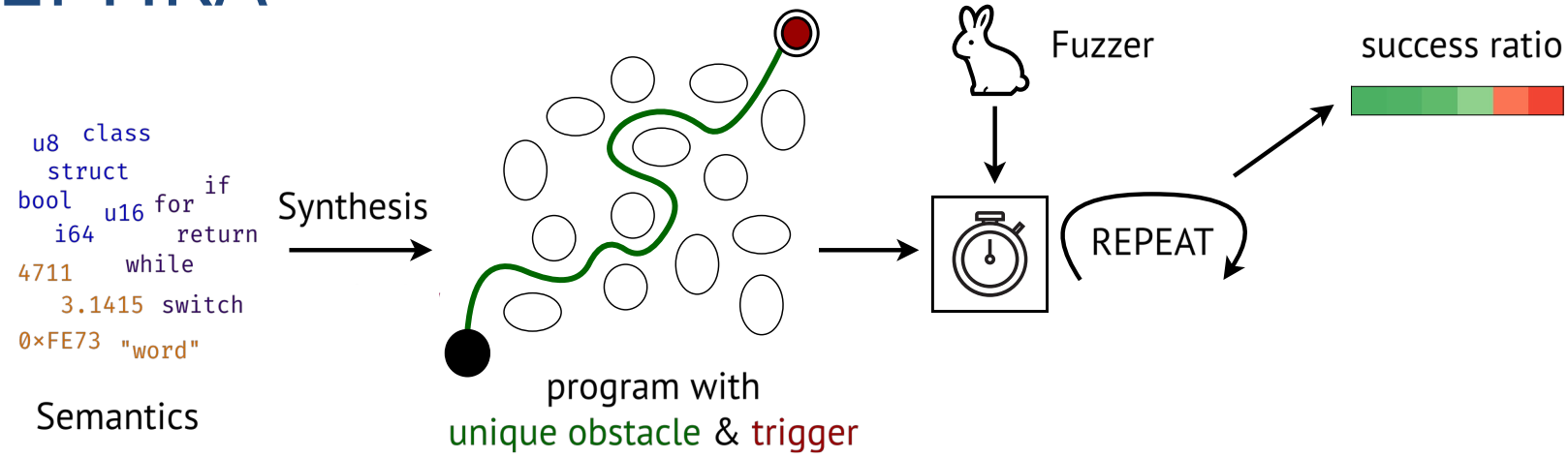
TEPHRA


u8 class
struct
bool u16 for if
i64 return
4711 while
3.1415 switch
0×FE73 "word"
Semantics
Synthesis
program with
unique obstacle & trigger
Fuzzer

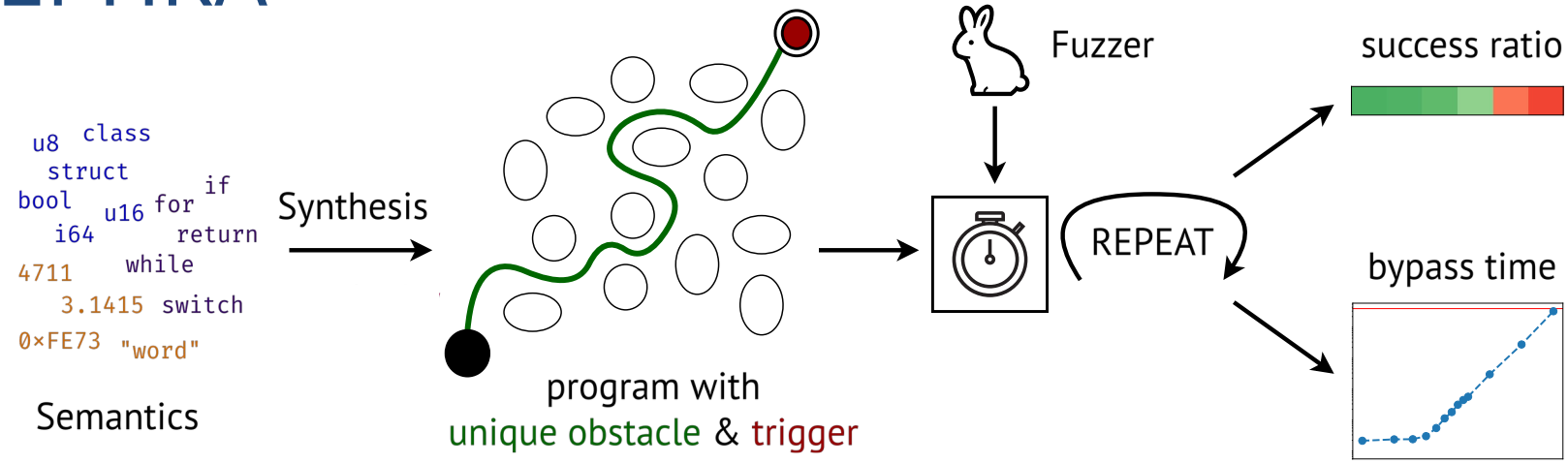
TEPHRA


u8 class
struct
bool u16 for if
i64 return
4711 while
3.1415 switch
0×FE73 "word"
Semantics
Synthesis
program with
unique obstacle & trigger
Fuzzer
REPEAT

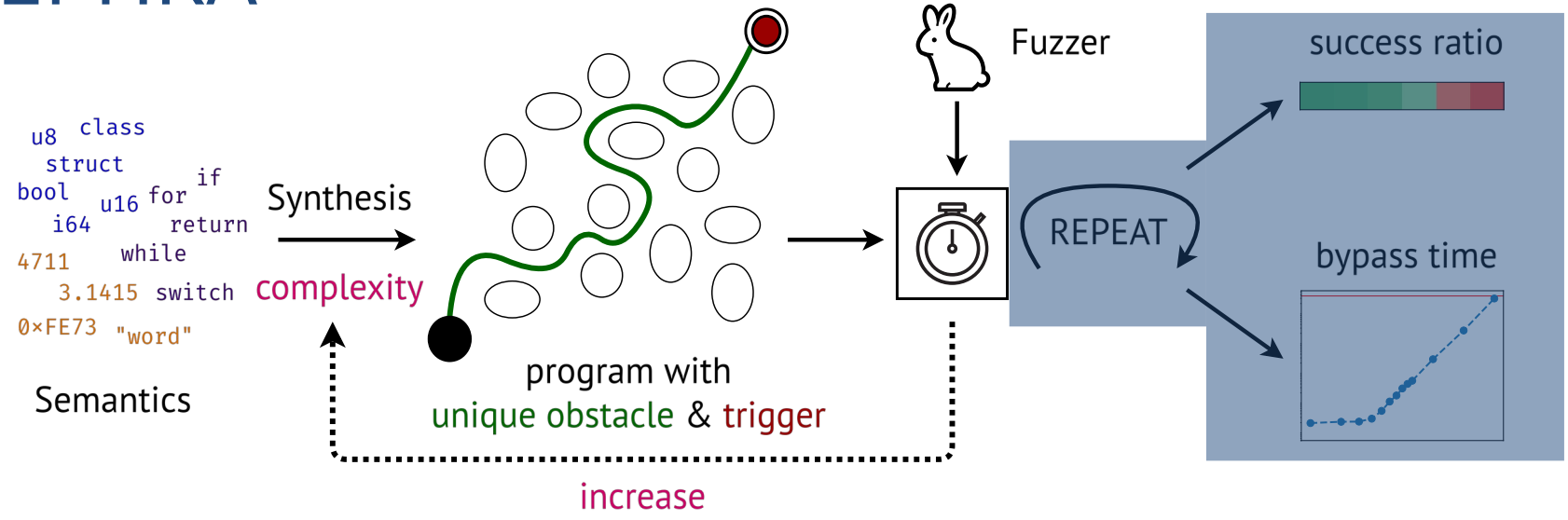
TEPHRA


u8 class
struct
bool u16 for if
i64 return
4711 while
3.1415 switch
0×FE73 "word"
Semantics
Synthesis
program with
unique obstacle & trigger
Fuzzer
REPEAT
success ratio

TEPHRA

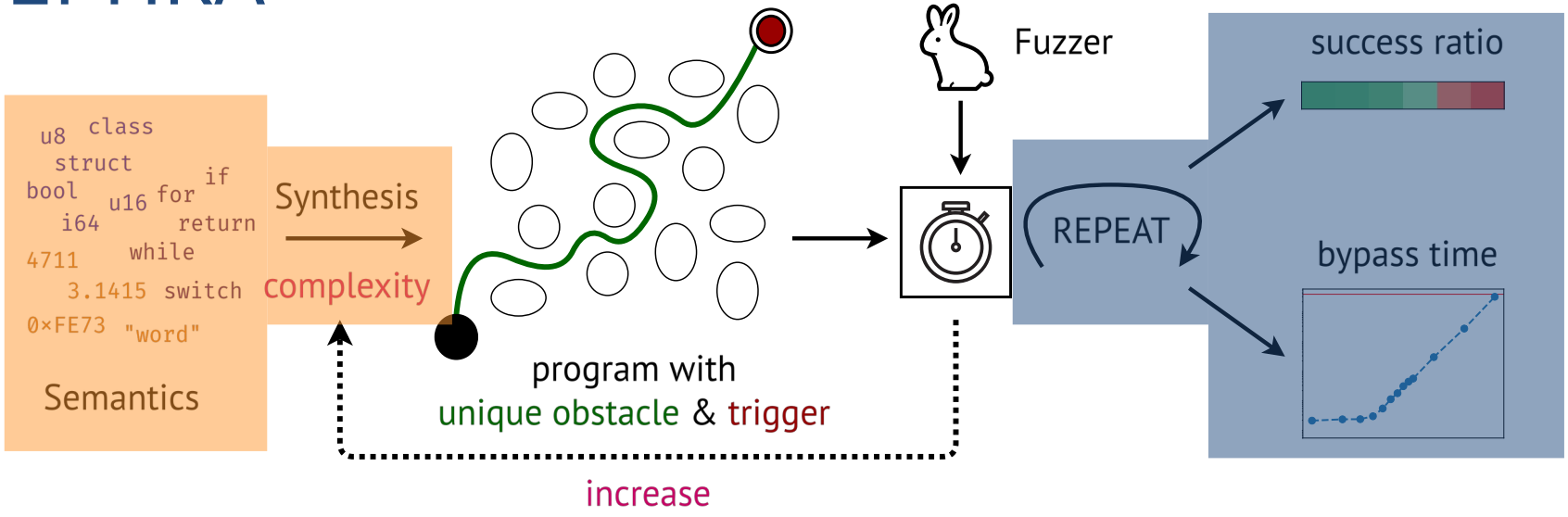

u8 class
struct
bool u16 for if
i64 return
4711 while
3.1415 switch
0×FE73 "word"
Semantics
Synthesis
program with
unique obstacle & trigger
Fuzzer
REPEAT
success ratio
bypass time

TEPHRA



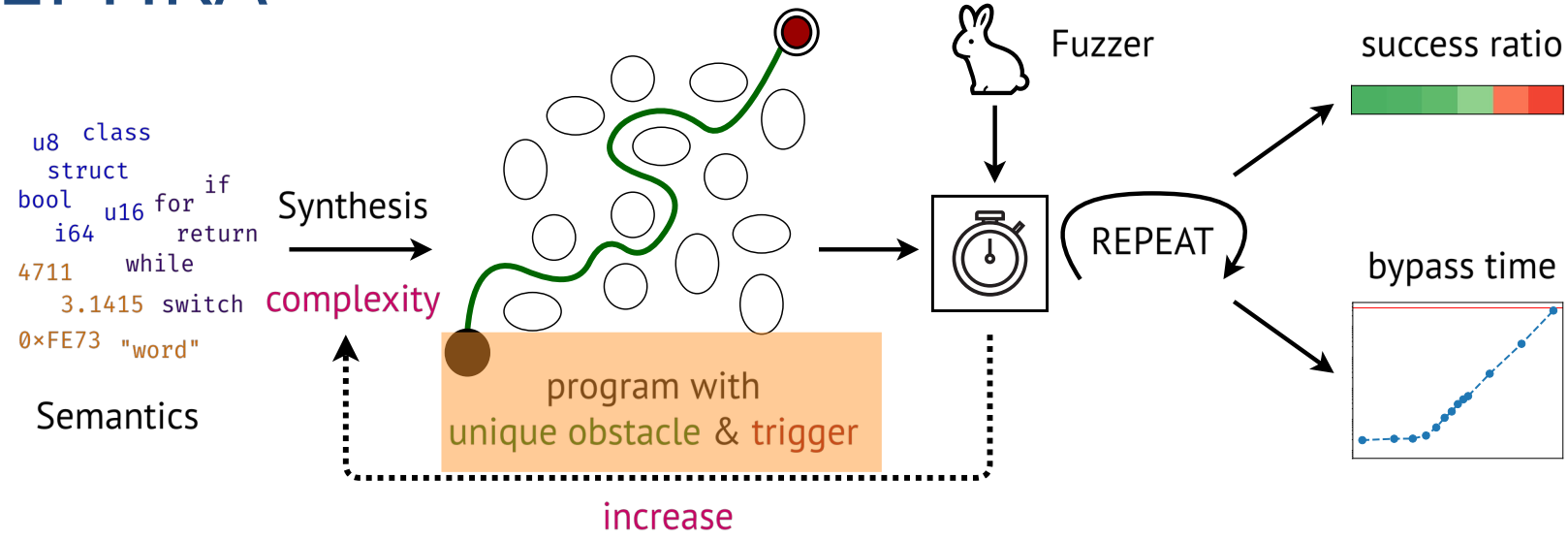
Principled Methodology = Analytical Model + Semantics-guided Program Synthesis

TEPHRA



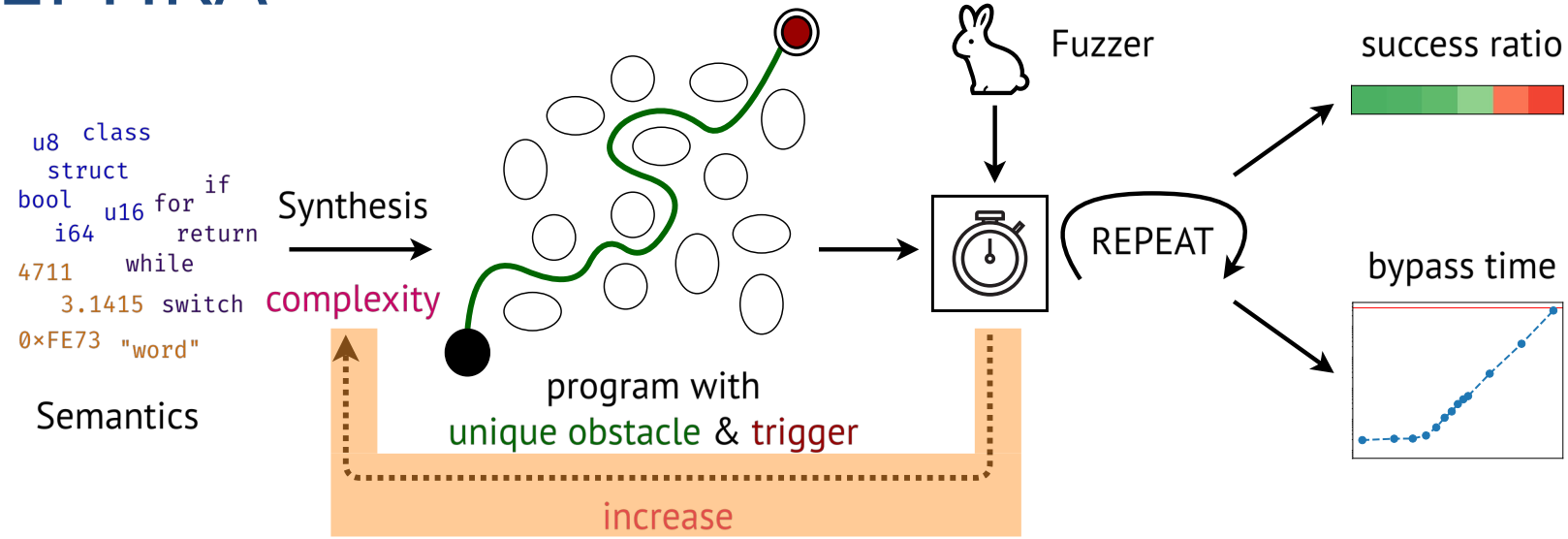
Principled Methodology = Analytical Model + Semantics-guided Program Synthesis

TEPHRA



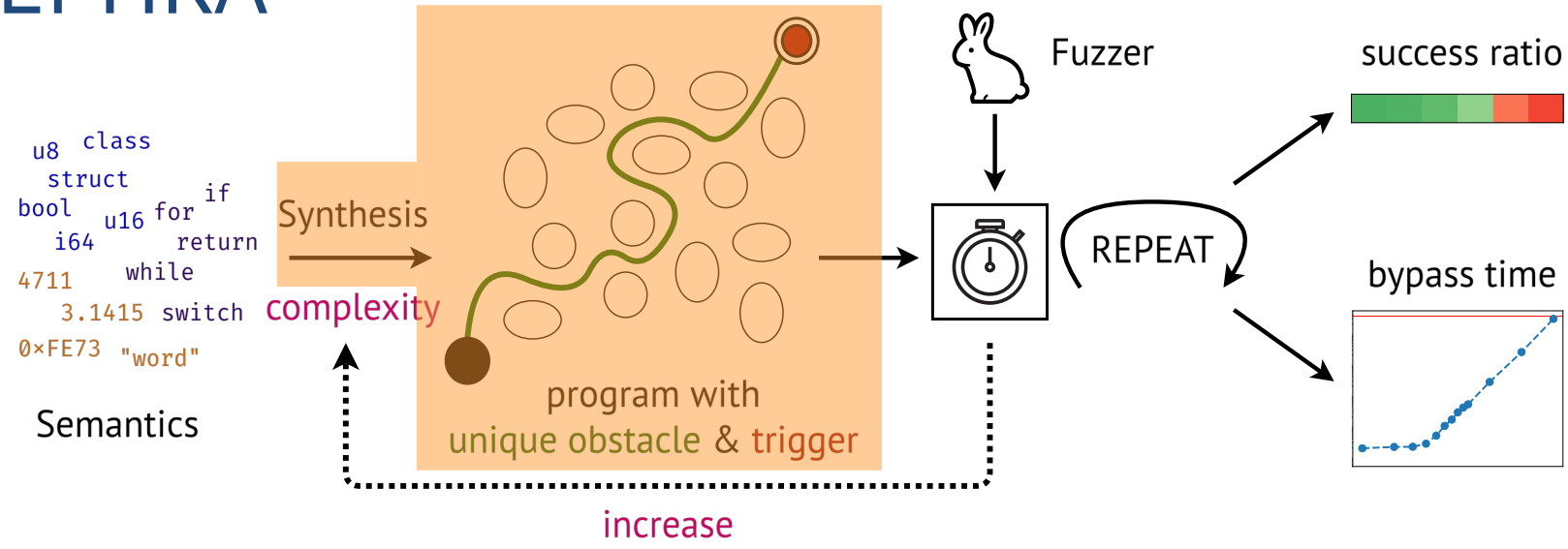
Yields *optimistic upper bounds*.

TEPHRA



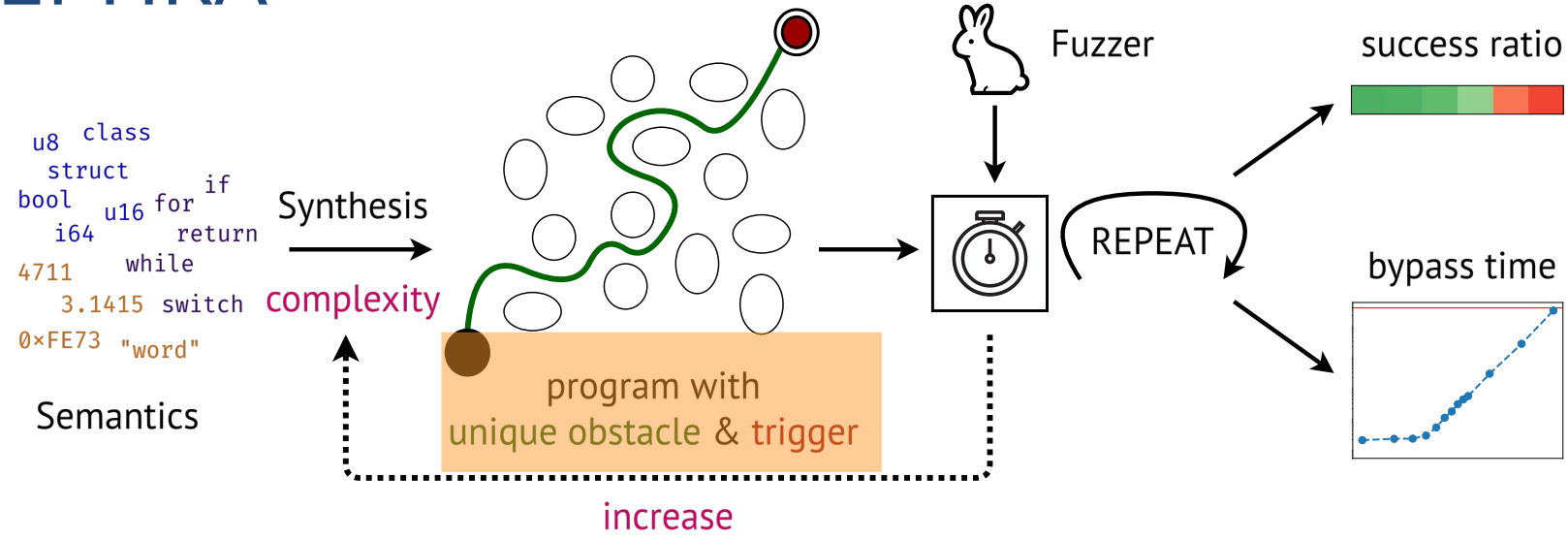
Systematically probes the semantic space

TEPHRA



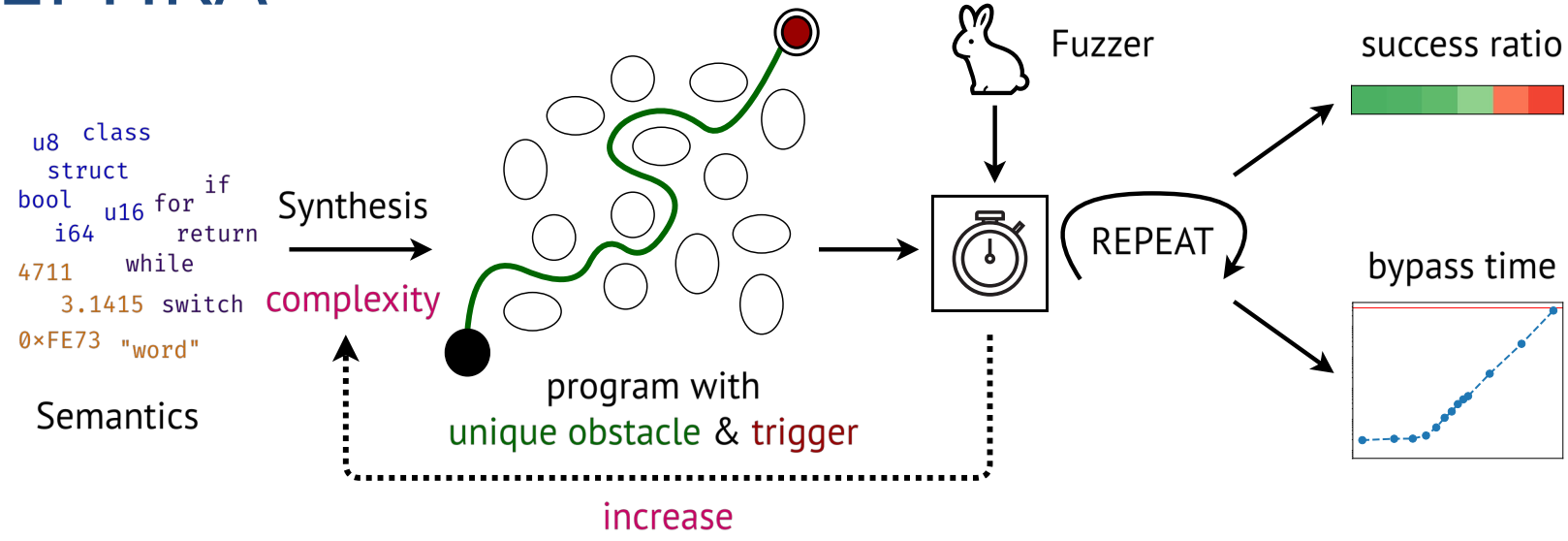
Synthesis is *pseudorandom* to *reduce* the risk of *overfitting*.

TEPHRA



Obstacle programs are *bug-free*.

TEPHRA

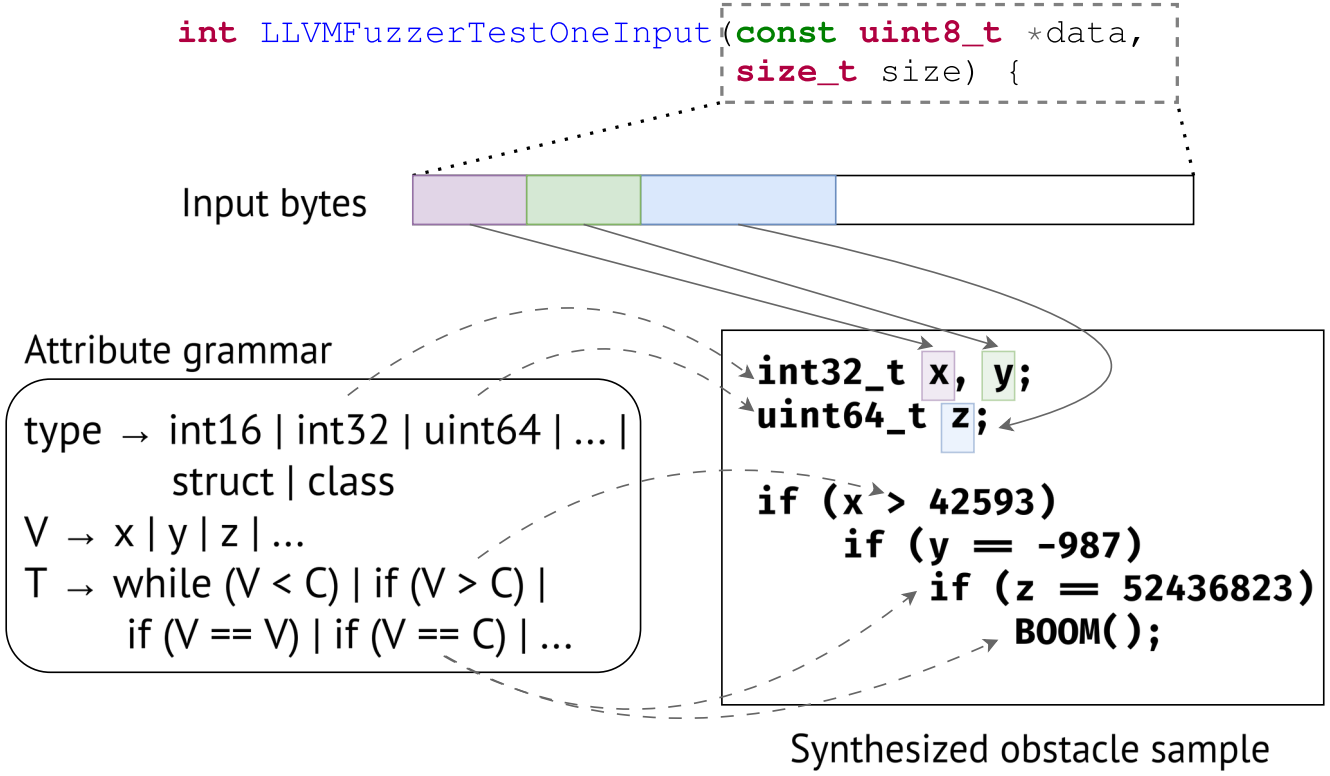


Complements existing fuzzing evaluation methods.

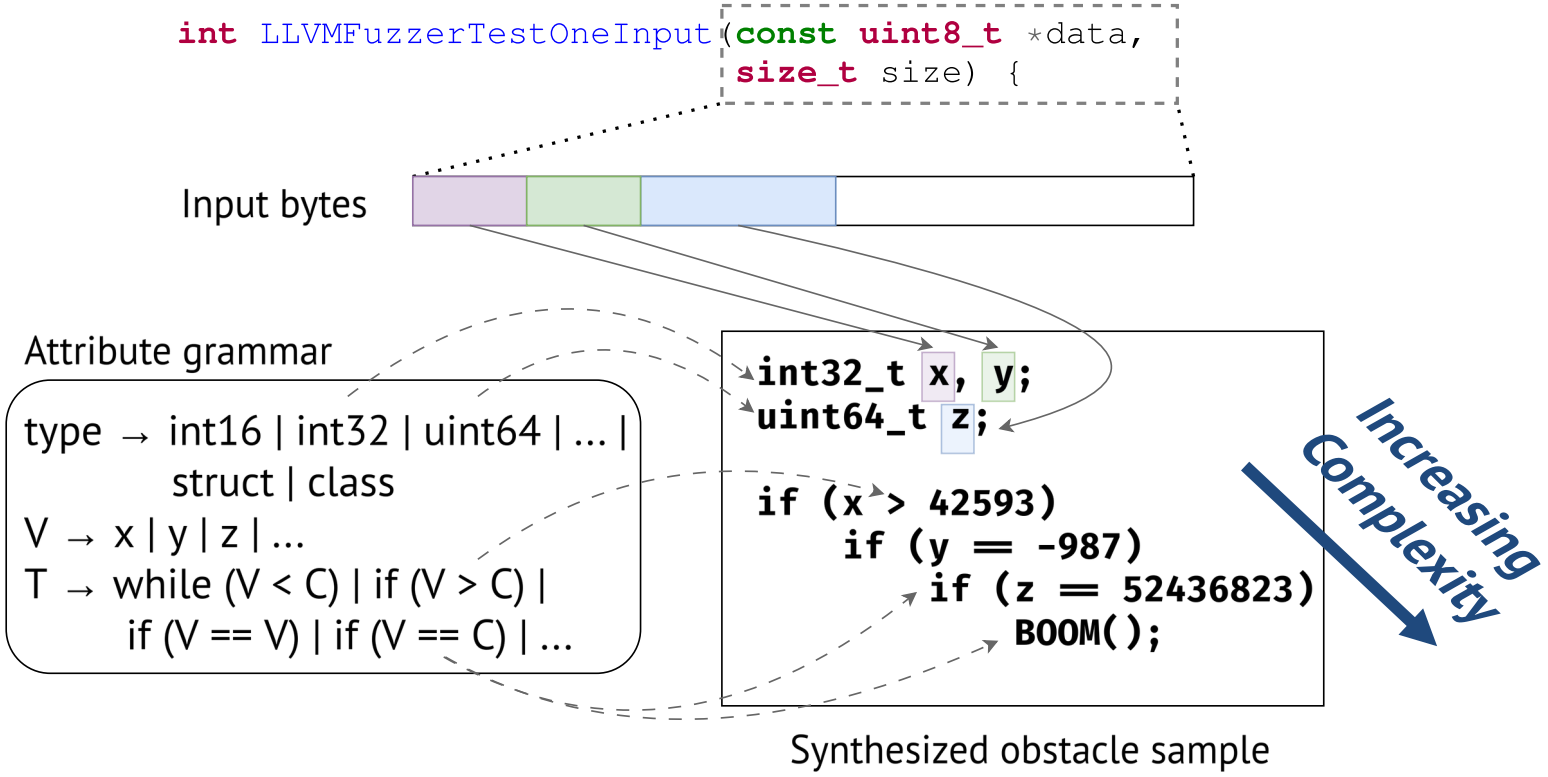
TEPHRA-C/C++

Implementation for C and C++

TEPHRA-C/C++: Synthesizing Obstacles



TEPHRA-C/C++: Synthesizing Obstacles



Empirical Study

Empirical Study

26 C/C++ semantic obstacles

Empirical Study

26 C/C++ semantic obstacles
31 Fuzzers

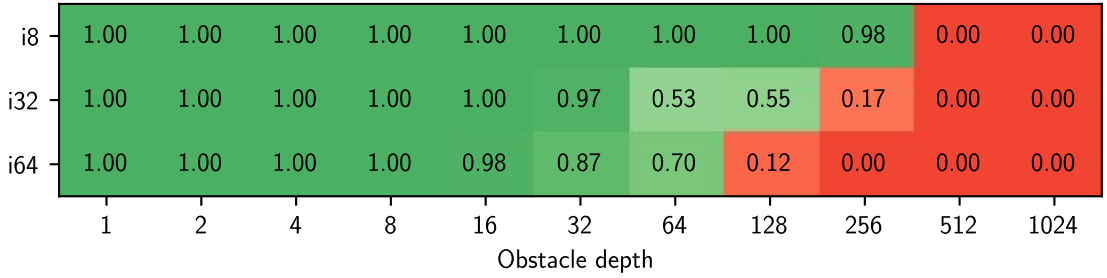
Empirical Study

26 C/C++ semantic obstacles
31 Fuzzers
37.4 CPU years

Results

Fuzzer	Chain													Interval 10^x							
		Unsigned Int				Signed Int				Float		Cmp		Unsigned Int			Signed Int			Float	
	bool	8	16	32	64	8	16	32	64	32	64	str	mem	16	32	64	16	32	64	32	64
AFL GCC	15	3	1	0	0	3	1	0	0	0	0	0	2	0	3	14	0	4	14	−4	−2
AFL clang-fast	14	15	1	0	0	14	1	0	0	0	0	0	2	0	3	14	0	4	14	−4	−2
AFL libtokencap	16	15	1	0	0	15	1	0	0	0	0	1	2	0	3	13	0	4	14	−4	−2
AFL QEMU	32	4	1	0	0	4	1	0	0	0	0	0	2	0	3	14	0	5	14	−4	−1
AFL++	15	15	2	64	32	14	5	3	32	0	0	5	32	0	3	13	0	4	12	−4	−3
AFL++ CmpLog	15	15	0	64	64	14	6	3	64	6	7	64	32	0	3	13	0	4	13	−6	−15
AFL++ laf-intel	15	13	0	16	1	13	1	3	1	1	1	4	16	0	1	11	0	0	11	−2	−1
AFL++ MOpt	15	14	0	32	0	14	5	3	32	0	0	0	32	0	3	13	0	3	12	−4	−3
AFL++ QEMU CmpLog	15	3	2	0	0	3	2	0	0	0	0	0	2	0	3	13	0	4	12	−6	−3
AFL++ QEMU CompCov	16	15	7	3	5	15	7	3	5	0	0	0	8	0	0	0	0	0	0	−4	−3
AFLGo	15	3	2	0	0	2	1	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
Angora	4096	8192	4096	2048	1024	8192	4096	2048	1024	0	0	0	2	0	1	0	0	0	0	−4	−3
Angora mb	8192	8192	4096	2048	1024	8192	4096	2048	1024	0	0	0	2	0	3	13	0	3	13	−4	−3
Angora random	8192	8192	10	0	0		14	0	0	0	0	0	2	0	4	13	0	4	12	−4	−3
DARWIN	512	64	1	0	0	64	3	0	0	0	0	0	2	0	3	13	0	3	12	−4	−4
datAFLow	14	2	1	0	0	2	1	0	0	0	0	0	2	0	3	13	0	4	14	−4	−3
DDFuzz	512	16	2	4	10	32	1	10	10	0	0	16	32	0	3	13	0	4	12	−4	−3
dev-urandom	13	3	1	0	0	3	1	0	0	0	0	0	3	0	2	12	0	2	12	−6	−4
EcoFuzz	128	14	1	9	0	15	7	9	0	0	0	0	4	0	0	7	0	0	6	−4	−3
FA-Fuzz	512	14	2	0	0	13	1	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
FairFuzz	1024	128	7	0	0	128	8	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
honggfuzz	256	32	32	16	16	32	32	32	16	0	0	32	32	0	0	0	0	0	0	−4	−1
honggfuzz QEMU	16	13	6	3	1		7	3	2	0	0	0	8	0	1	13	0	0	8	−4	−1
KLEE	128	128	64	32	16	128	64	32	16	0	0	16	128	0	0	0	0	0	0	0	0
LibAFL	16	15	7	3	16	15	7	10	16	0	0	5	32	0	3	12	0	2	11	−5	−5
LibAFL_libFuzzer	512	64	5	0	1024	64	3	1024	1024	0	0	0	3	0	0	0	0	0	0	−5	−5
libFuzzer	512	256	64	16	128	128	32	64	32	0	0	64	64	0	0	0	0	0	0	−4	−3
libFuzzer Entropic	512	256	32	16	64	256	32	128	64	0	0	64	64	0	0	0	0	0	0	−5	−3
Radamsa	13	2	1	0	0	2	1	0	0	0	0	0	2	–	4	13	–	4	14	−4	–
SymCC	13	9	9	9	10	9	9	9	10	0	0	0	16384	0	0	0	0	0	0	0	0
WingFuzz	1024	256	128	32	128	256	32	256	128	0	0	64	64	0	0	0	0	0	0	−5	−3

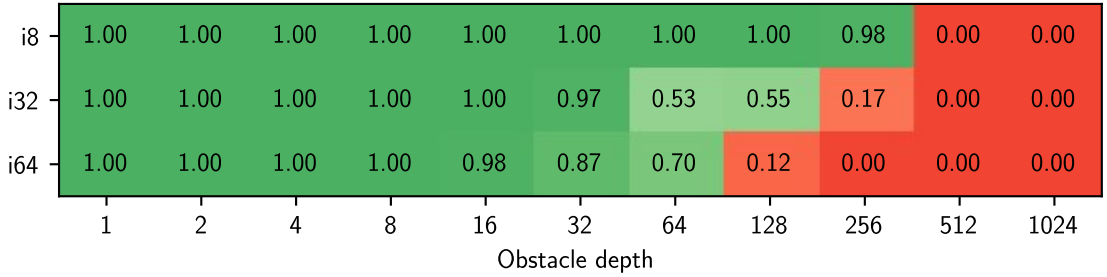
Results: Hunting for



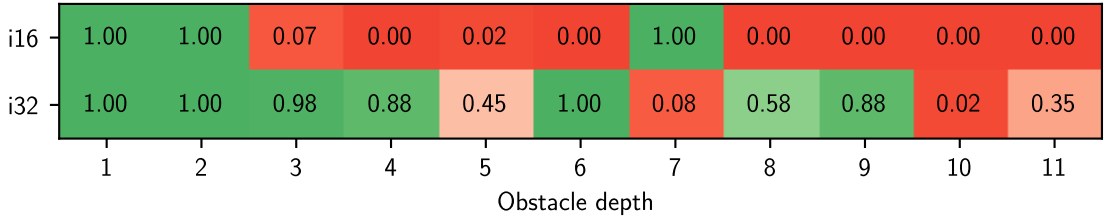
Stable Behavior

Increasing Complexity

Results: Hunting for **Anomalies**

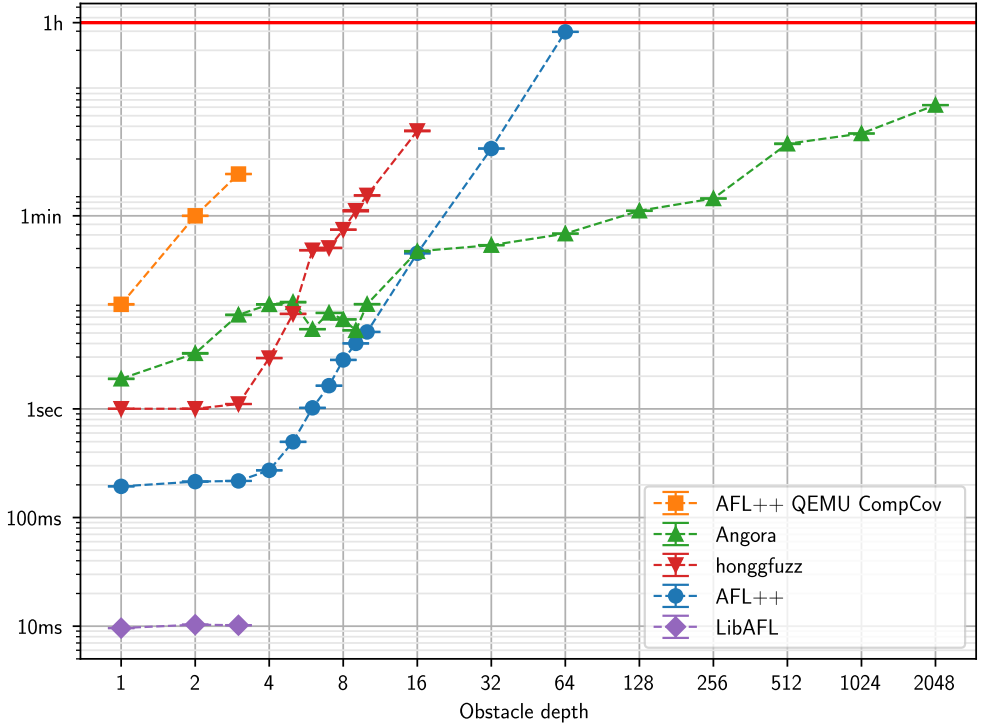


Stable vs. Unstable Behavior

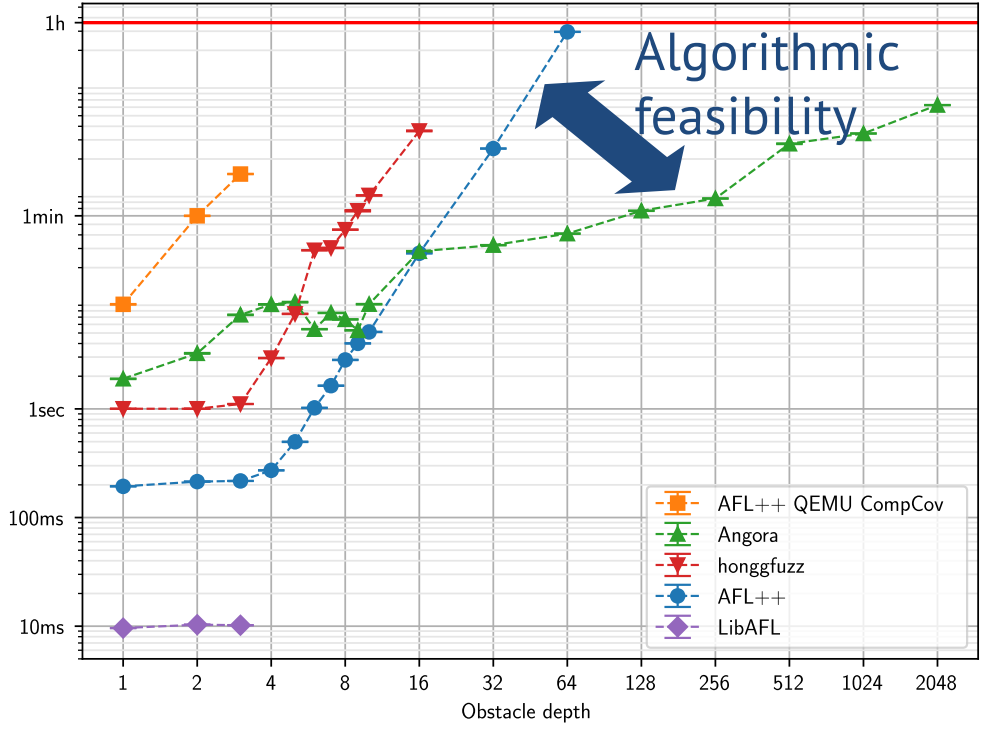


Increasing Complexity

Results: Performance Differences



Results: Performance Differences



Results: Summary

- All fuzzers struggle with certain semantic constructs.
- Support for rational numbers and character strings lacking.
- Signed integers more difficult than unsigned.
- Overtuning for 32- and 64-bit types, neglecting 8- and 16-bit.
- No fuzzer excels across all obstacles.
- A single obstacle can degrade overall performance.
- ~90% stable obstacle bypasses.
- There are bugs in the fuzzers themselves.

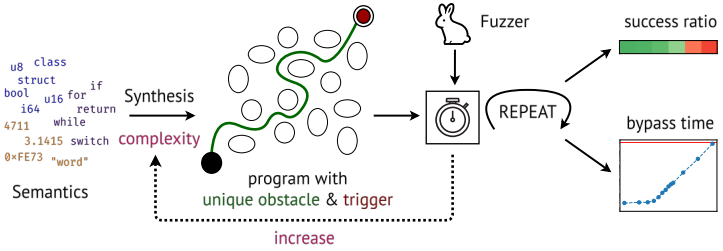
Fuzzer	Chain													Interval 10^x							
		Unsigned Int				Signed Int				Float		Cmp		Unsigned Int			Signed Int			Float	
	bool	8	16	32	64	8	16	32	64	32	64	str	mem	16	32	64	16	32	64	32	64
AFL GCC	15	3	1	0	0	3	1	0	0	0	0	0	2	0	3	14	0	4	14	−4	−2
AFL clang-fast	14	15	1	0	0	14	1	0	0	0	0	0	2	0	3	14	0	4	14	−4	−2
AFL libtokencap	16	15	1	0	0	15	1	0	0	0	0	1	2	0	3	13	0	4	14	−4	−2
AFL QEMU	32	4	1	0	0	4	1	0	0	0	0	0	2	0	3	14	0	5	14	−4	−1
AFL++	15	15	2	64	32	14	5	3	32	0	0	5	32	0	3	13	0	4	12	−4	−3
AFL++ CmpLog	15	15	0	64	64	14	6	3	64	6	7	64	32	0	3	13	0	4	13	−6	−15
AFL++ laf-intel	15	13	0	16	1	13	1	3	1	1	1	4	16	0	1	11	0	0	11	−2	−1
AFL++ MOpt	15	14	0	32	0	14	5	3	32	0	0	0	32	0	3	13	0	3	12	−4	−3
AFL++ QEMU CmpLog	15	3	2	0	0	3	2	0	0	0	0	0	2	0	3	13	0	4	12	−6	−3
AFL++ QEMU CompCov	16	15	7	3	5	15	7	3	5	0	0	0	8	0	0	0	0	0	0	−4	−3
AFLGo	15	3	2	0	0	2	1	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
Angora	4096	8192	4096	2048	1024	8192	4096	2048	1024	0	0	0	2	0	1	0	0	0	0	−4	−3
Angora mb	8192	8192	4096	2048	1024	8192	4096	2048	1024	0	0	0	2	0	3	13	0	3	13	−4	−3
Angora random	8192	8192	10	0	0		14	0	0	0	0	0	2	0	4	13	0	4	12	−4	−3
DARWIN	512	64	1	0	0	64	3	0	0	0	0	0	2	0	3	13	0	3	12	−4	−4
datAFLow	14	2	1	0	0	2	1	0	0	0	0	0	2	0	3	13	0	4	14	−4	−3
DDFuzz	512	16	2	4	10	32	1	10	10	0	0	16	32	0	3	13	0	4	12	−4	−3
dev-urandom	13	3	1	0	0	3	1	0	0	0	0	0	3	0	2	12	0	2	12	−6	−4
EcoFuzz	128	14	1	9	0	15	7	9	0	0	0	0	4	0	0	7	0	0	6	−4	−3
FA-Fuzz	512	14	2	0	0	13	1	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
FairFuzz	1024	128	7	0	0	128	8	0	0	0	0	0	2	0	3	13	0	4	13	−4	−3
honggfuzz	256	32	32	16	16	32	32	32	16	0	0	32	32	0	0	0	0	0	0	−4	−1
honggfuzz QEMU	16	13	6	3	1		7	3	2	0	0	0	8	0	1	13	0	0	8	−4	−1
KLEE	128	128	64	32	16	128	64	32	16	0	0	16	128	0	0	0	0	0	0	0	0
LibAFL	16	15	7	3	16	15	7	10	16	0	0	5	32	0	3	12	0	2	11	−5	−5
LibAFL_libFuzzer	512	64	5	0	1024	64	3	1024	1024	0	0	0	3	0	0	0	0	0	0	−5	−5
libFuzzer	512	256	64	16	128	128	32	64	32	0	0	64	64	0	0	0	0	0	0	−4	−3
libFuzzer Entropic	512	256	32	16	64	256	32	128	64	0	0	64	64	0	0	0	0	0	0	−5	−3
Radamsa	13	2	1	0	0	2	1	0	0	0	0	0	2	–	4	13	–	4	14	−4	–
SymCC	13	9	9	9	10	9	9	9	10	0	0	0	16384	0	0	0	0	0	0	0	0
WingFuzz	1024	256	128	32	128	256	32	256	128	0	0	64	64	0	0	0	0	0	0	−5	−3

Results: Byproducts

```
clang version 20.1.0
...
PLEASE submit a bug report to https://github.com/llvm/llvm-project/issues/
and include the crash backtrace, preprocessed source, and associated run script.
Stack dump:
...
1.      bool_chain_exp/tc_8192.c:29661:27: current parser token ')'
2.      bool_chain_exp/tc_8192.c:9:58: parsing function body
'LLVMFuzzerTestOneInput'
3.      bool_chain_exp/tc_8192.c:9:58: in compound statement ('{}')
clang: error: unable to execute command: Segmentation fault
clang: error: clang frontend command failed due to signal (use -v to see invocation)
```

Summary and Conclusion

- Contributions
  - **TEPHRA**: *principled methodology* to empirically discover fuzzer limitations.
  - **TEPHRA-C/C++**: implementation for C/C++.
  - Initial study: counterintuitive limitations found.
- Next Steps
  - Further experimentation.
  - Extend semantics for C/C++.
  - Implementation for other PLs.
  - Fuzzing based on obstacle profiles.



ucsrl.de/research/tephra

Find us at the poster session!