TEPHRA

Principled Discovery of Fuzzer Limitations

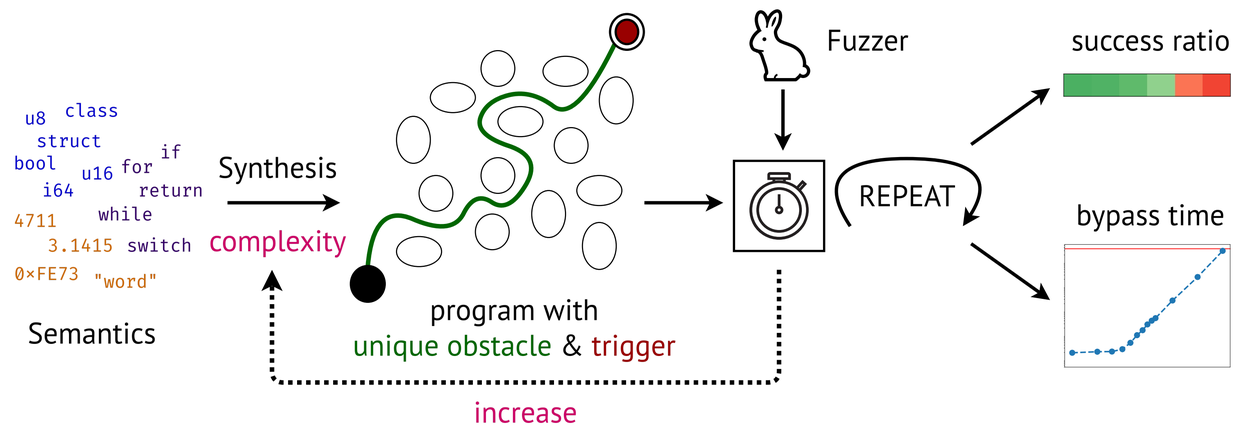
Vasil Sarafov David Markvica Stefan Brunthaler

Fuzzers have limitations

- Bug finding is undecidable, fuzzers rely on heuristics.
- How hard is “too hard” for a fuzzer?
- Which heuristics work best for specific obstacles?

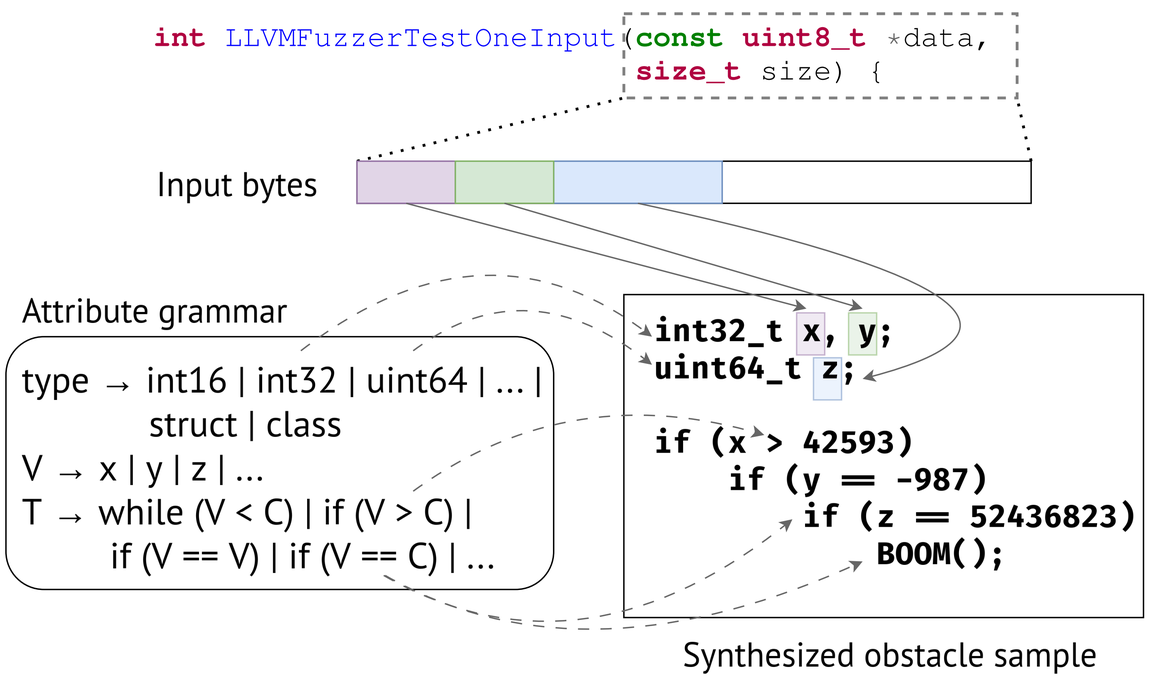
How to identify fuzzer limitations?

- TEPHRA: principled methodology that yields optimistic upper bounds.
- Reduces overfitting & complements existing evaluation approaches.
- Systematically probes the semantic space.



TEPHRA

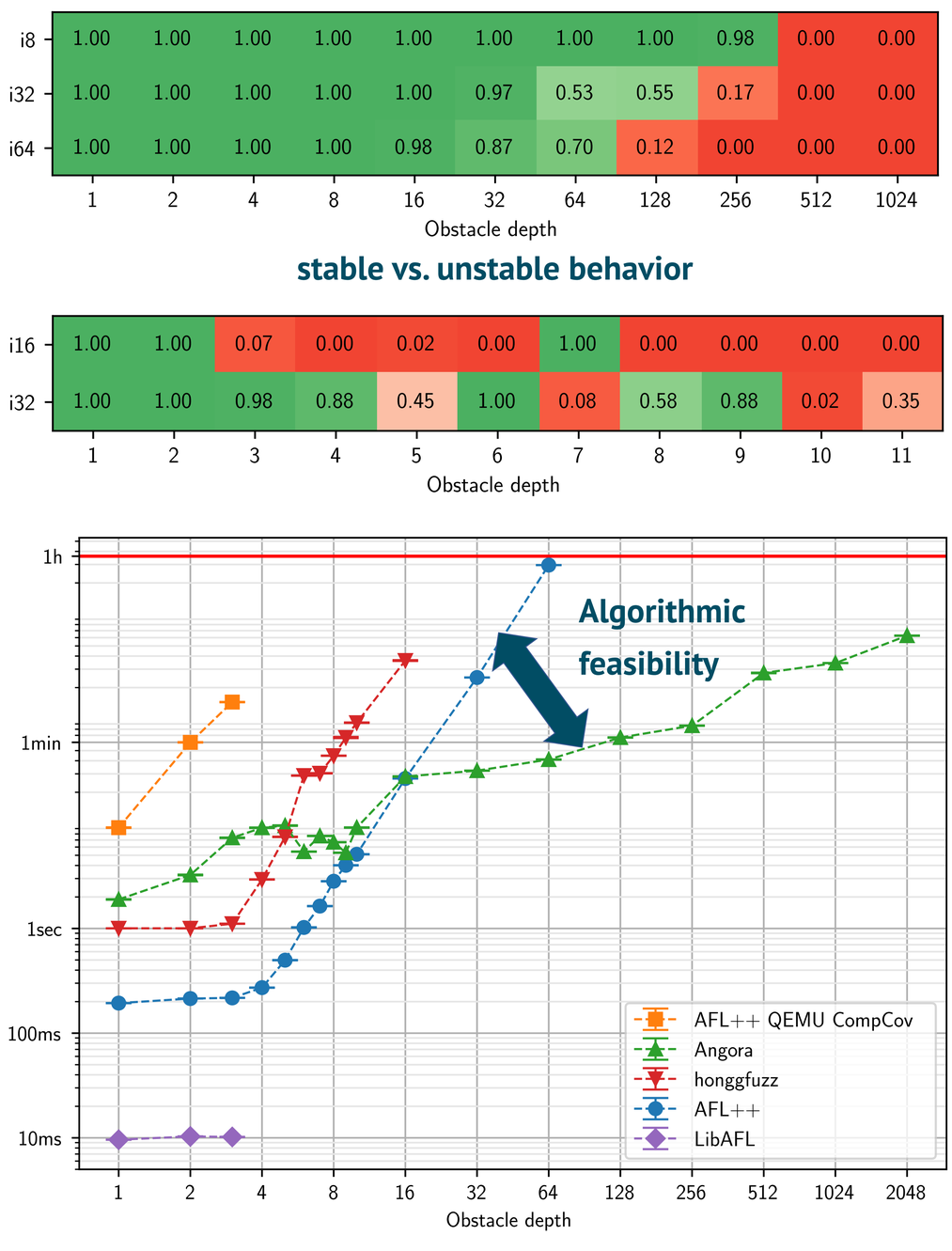
TEPHRA-C/C++




```
int LLVMFuzzerTestOneInput(const uint8_t *data,
                           size_t size) {
    uint16_t x, y, z, u;
    if (size < 8)
        return TEPHRA_EXIT_FAILURE;

    memcpy(&x, &data[0], 2);
    memcpy(&y, &data[2], 2);
    memcpy(&z, &data[4], 2);
    memcpy(&u, &data[6], 2);

    if (x == 38951)
        if (y == 13747)
            if (z == 54130)
                if (u == 7810)
                    BOOM();

    return TEPHRA_EXIT_SUCCESS;
}
```

Increasing Complexity

31 fuzzers, 26 C/C++ semantic obstacles, 37.4 CPU years



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

- 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% reliable bypasses; ~10% due to fluctuating randomness.

Next Steps

- Further experiments.
- Fuzzing based on an obstacle profile.
- Extend implementation with additional semantics and PLs.

ucsrl.de/research/tephra