Fuzzing

Technique:

Try to crash f

by picking the right x

```
fn f(x: int) {
  let y = x + 2 in
  if (y > 10) {
    assert(x < 100)
  }
}
```

correctness conditions

Option #1: enumerate all x

run f on all x

for k steps

for k minutes

mostly

Pros: exhaustive

Cons: time intensive

impossible if input space is infinite

fuzzer

Option #2: run on random inputs

^ don't repeat on prior inputs

Pros: input space coverage

if lucky, extra good

Cons: consistency

if unlucky, extra bad

Option #3: run on known bad inputs

Option #4: A) stratified sampling

B) control flow coverage

Pros: pick better samples w/ feedback

Cons: ① restrict test expressiveness

② coverage might not correspond to bugs

③ overhead in time & space

Coverage-guided fuzzing

```
fn example(s: &str) {
0:  if s[0] == "a" {
1:      if s[1] == "b" {
2:          assert(s[2] == "c");
        }
    }
}
```

seeds: {"", "a quick fox", "∅"}

{0} {0, 1} {0}

"abquick" "aquick"

{0, 1, 2} {0, 1}

0 → 1 → 2 0 → 1

```
fuzz(f):
    pool = {"", ...}
    while time left:
        for x ∈ pool:
            execute f(x) for k seconds
            collect coverage information
            if coverage ⊄ ∪ all existing coverage:
                add mutations(x) to pool'
        pool = pool'
```