Program Testing
Hybrid Fuzzing

Yue Duan

Outline

- Hybrid fuzzing recap
- Research paper:
  - Driller: Augmenting Fuzzing through Symbolic Execution
  - Send Hardest Problems My Way: Probabilistic Path Prioritization for Hybrid Fuzzing

Grey-box Fuzzing

```
x = int(input())
if x > 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Let's fuzz it!

1 ⇒ "You lose!"

593 ⇒ "You lose!"

183 ⇒ "You lose!"

4 ⇒ "You lose!"

498 ⇒ "You lose!"

48 ⇒ "You win!"

Grey-box Fuzzing

```
x = int(input())
if x > 10:
    if x^2 == 152399025:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Let's fuzz it!

1 ⇒ "You lose!"

593 ⇒ "You lose!"

183 ⇒ "You lose!"

4 ⇒ "You lose!"

498 ⇒ "You lose!"

42 ⇒ "You lose!"

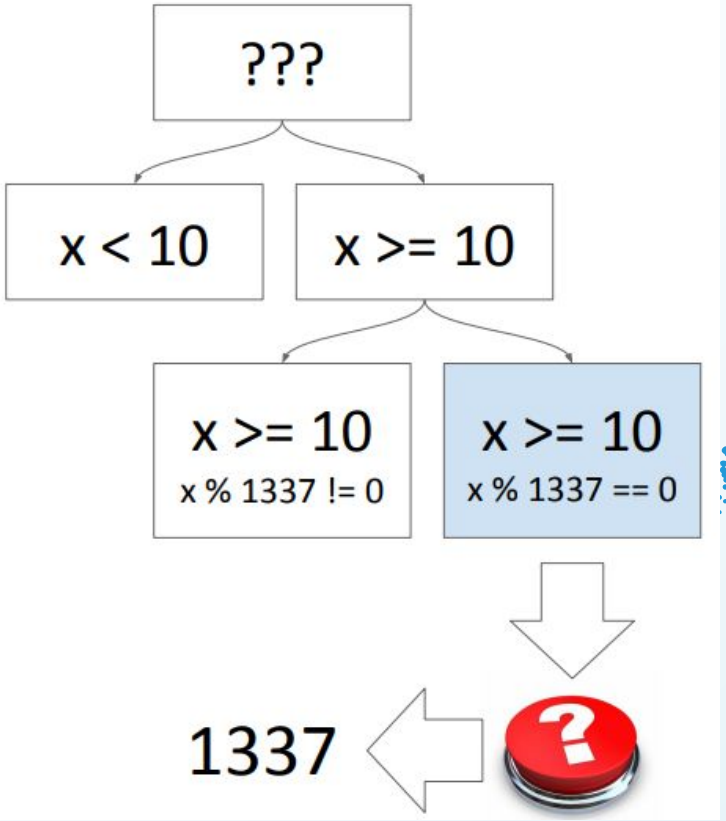
3 ⇒ "You lose!"

.....

57 ⇒ "You lose!"

Symbolic Execution

```
x = input()
if x >= 10:
    if x % 1337 == 0:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Hybrid Fuzzing

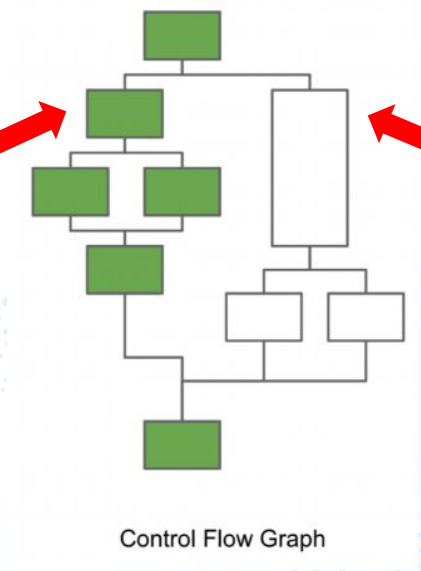
- Grey-box fuzzing
  - fast: throughput > 1000
  - good at exploring general branches
    - (e.g., x > 1) branches that have large satisfying value spaces
  - bad at exploring specific branches
    - (e.g., x * 2 == 343212) branches that have small satisfying value spaces
- Symbolic Execution
  - slow: multiple mins for one execution
    - path explosion
  - can generate concrete inputs that lead to specific paths

Hybrid Fuzzing

- Key idea:
  - let fuzzer take major responsibility for exploration
    - take advantage of its high efficiency
  - let symbolic execution solve hard problems
    - utilize its capability of solving specific conditions
    - avoid path explosion

Hybrid Fuzzing

Condition: if (a > 5)



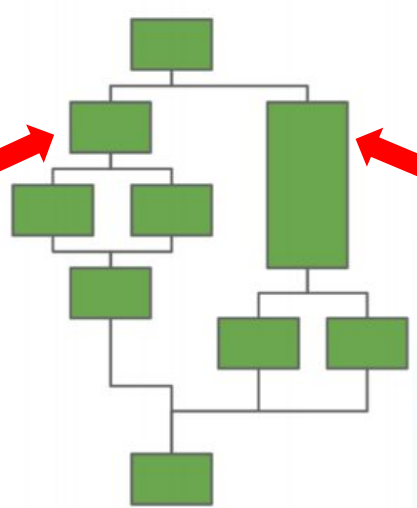
Control Flow Graph

Condition: if (a == 0x43135)

Unreachable by fuzzing

Hybrid Fuzzing

Fuzzer:
a = 6



Control Flow Graph

Symbolic execution:
a = 0x43135

Major Challenge

How to **distribute** the workload?

Driller: Augmenting Fuzzing through Symbolic Execution

Nick Stephens, John Grosen, Christopher Salls, Audrey Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna

NDSS 2016

Fuzzing vs. Symbolic Execution

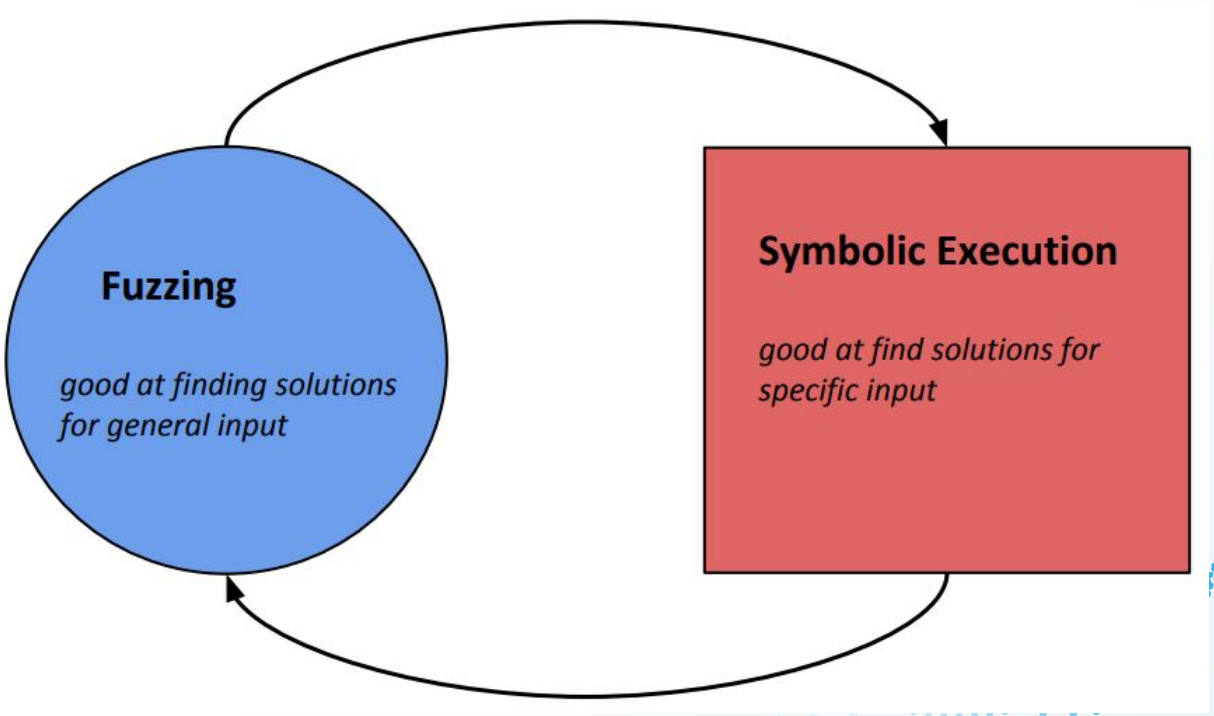
```
x = input()

def recurse(x, depth):
  if depth == 2000
    return 0
  else {
    r = 0;
    if x[depth] == "B":
      r = 1
    return r + recurse(x
[depth], depth)

if recurse(x, 0) == 1:
  print "You win!"
```

Fuzzing Wins

```
x = int(input())
if x >= 10:
    if x^2 == 152399025:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Symbolic Execution Wins

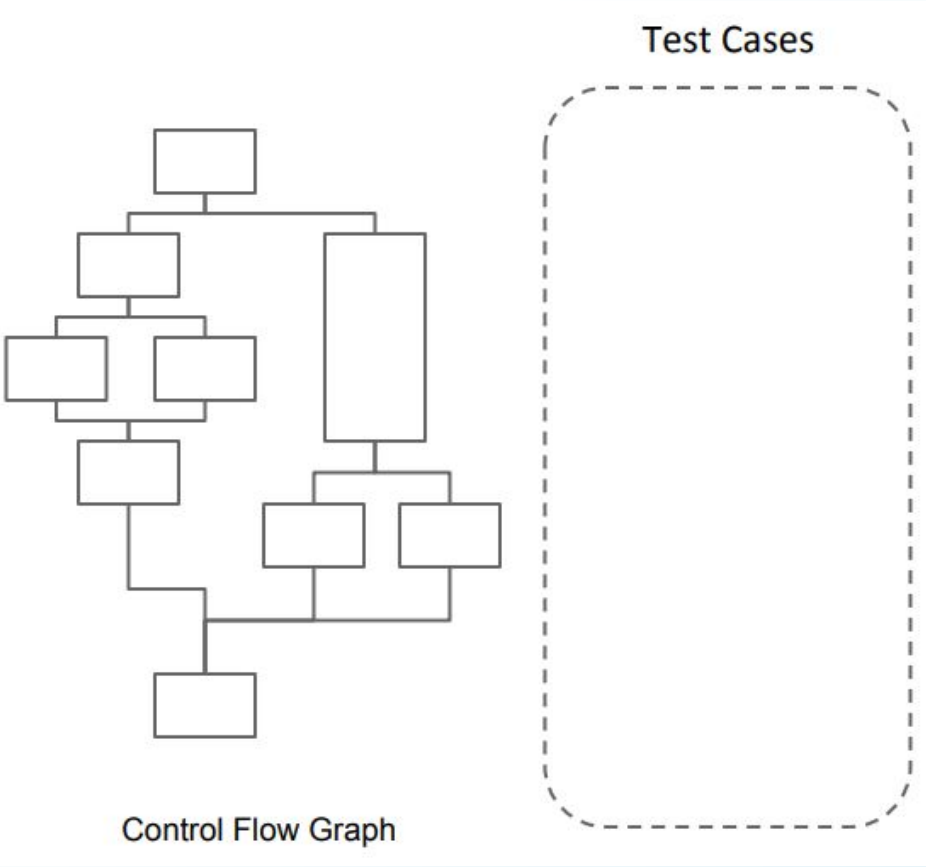
Introduction



Introduction

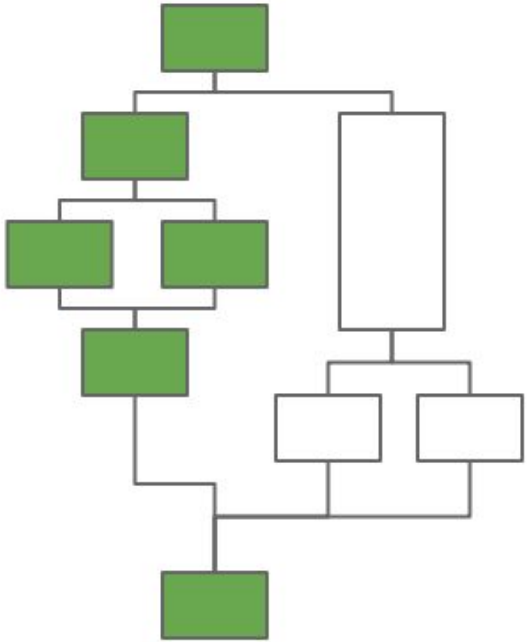
- Fuzzing + Symbolic Execution
  - AFL
    - state-of-the-art grey-box fuzzer
    - random mutation
    - basic block coverage tracking
  - Angr
    - symbolic execution engine
    - works on binary code

Combining the Two



Combining the Two

"Cheap" fuzzing coverage



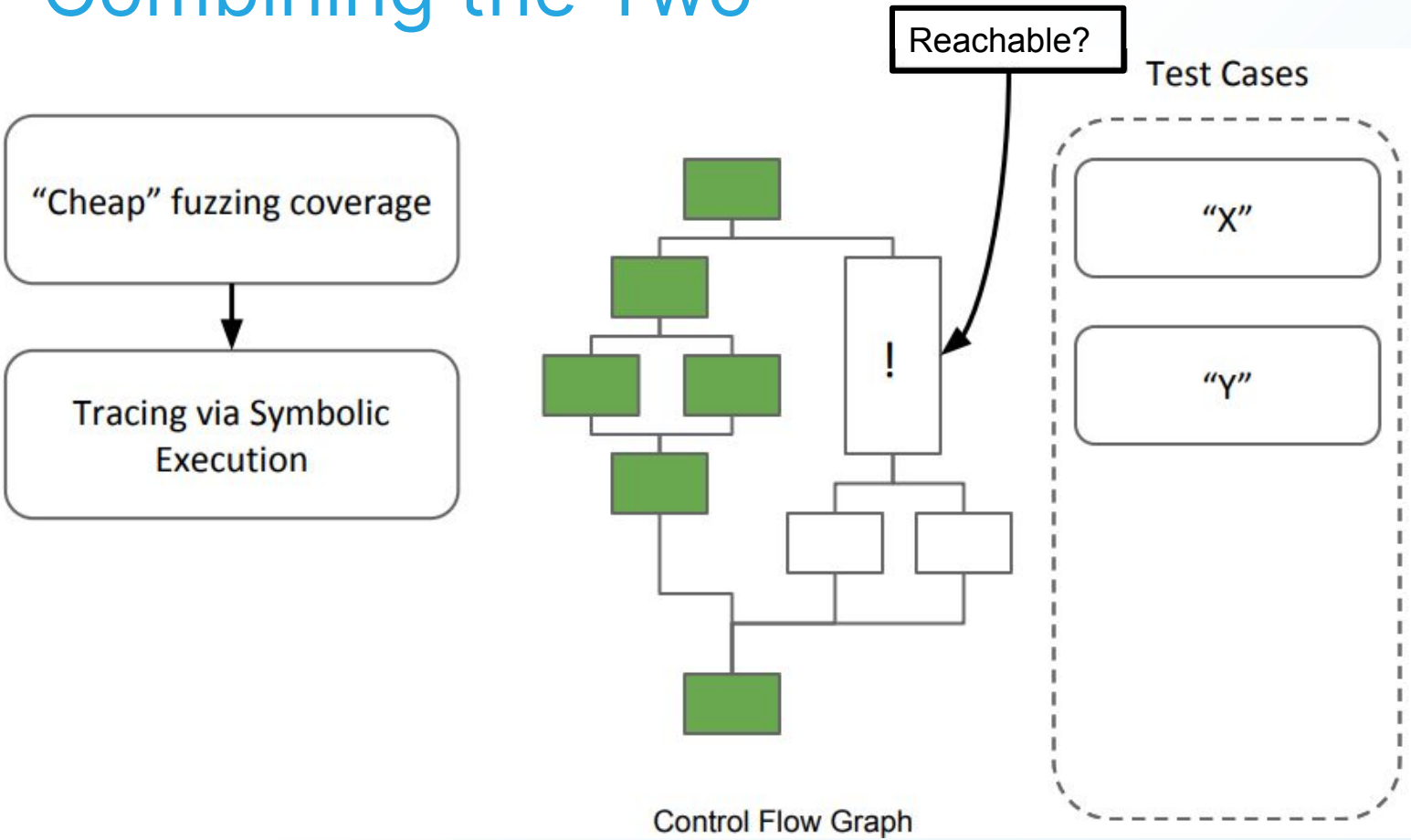
Control Flow Graph

Test Cases

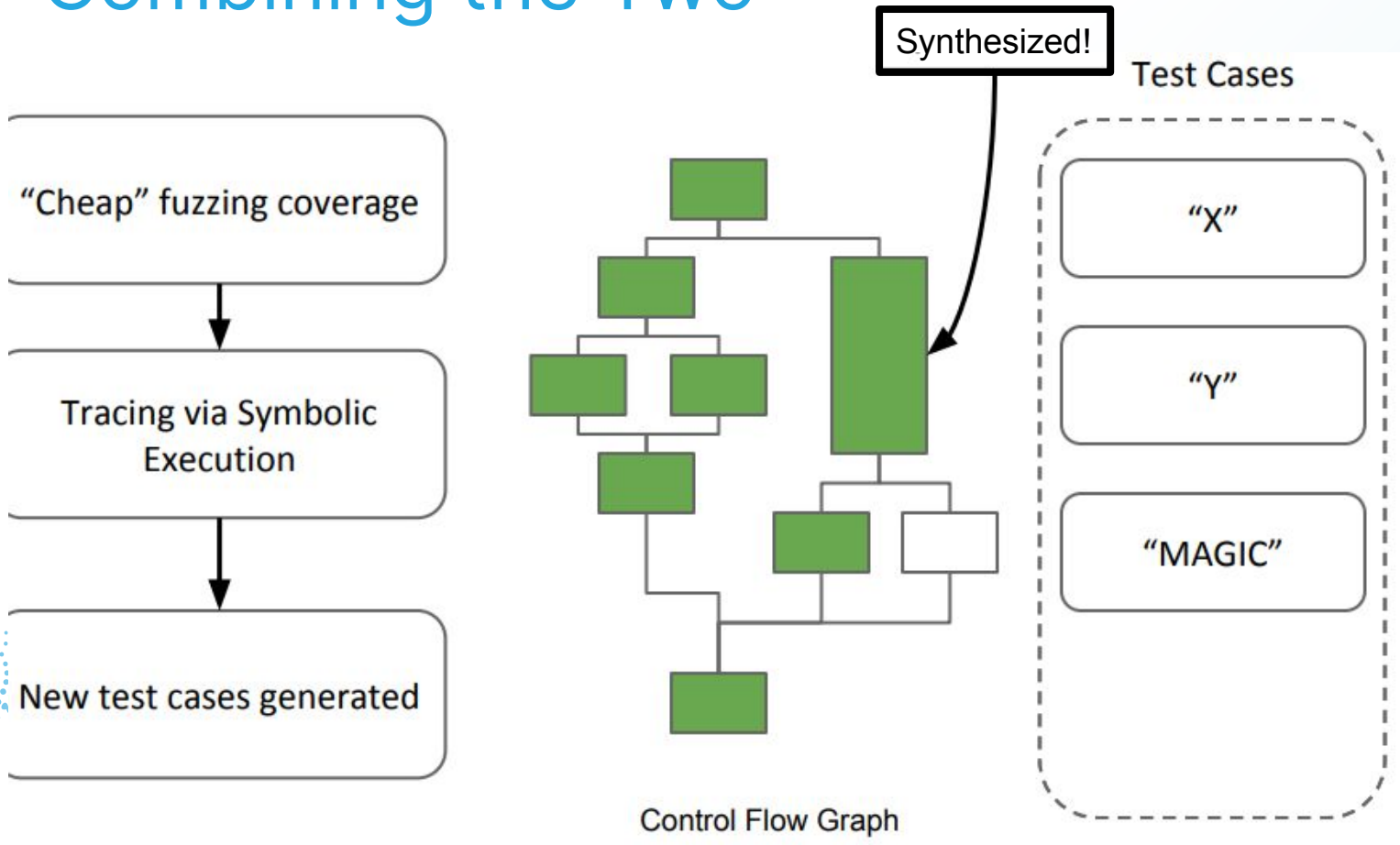
"X"

"Y"

Combining the Two

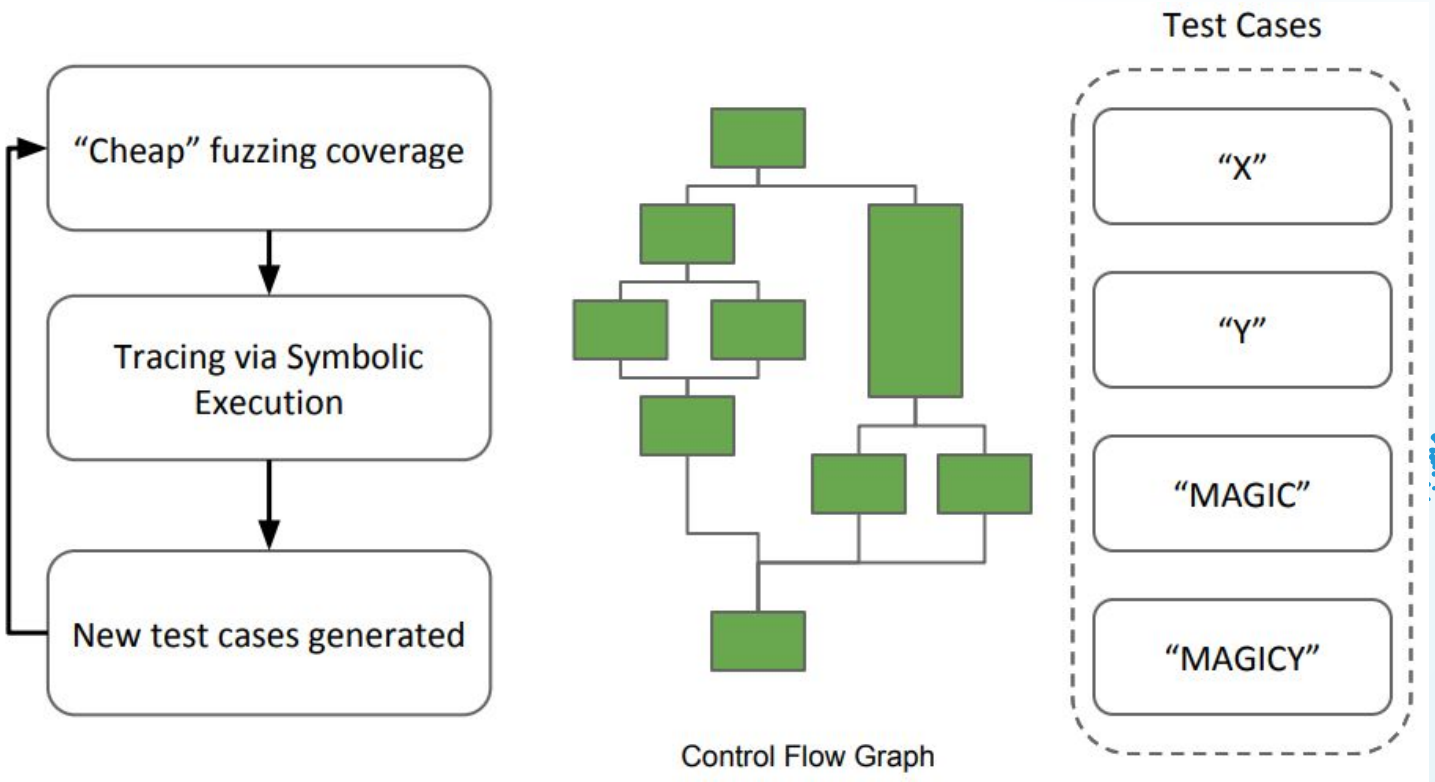


Combining the Two



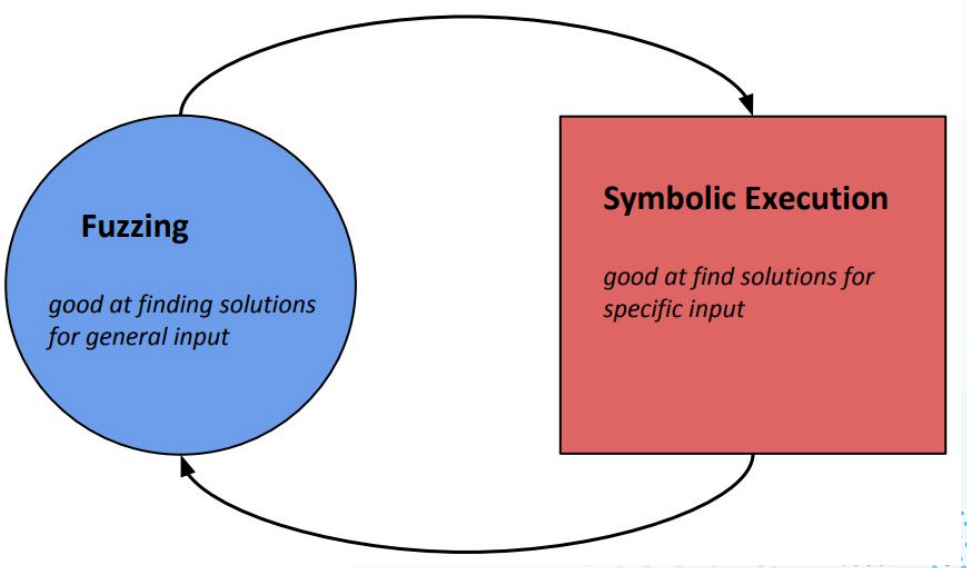
Combining the Two

Towards better code coverage



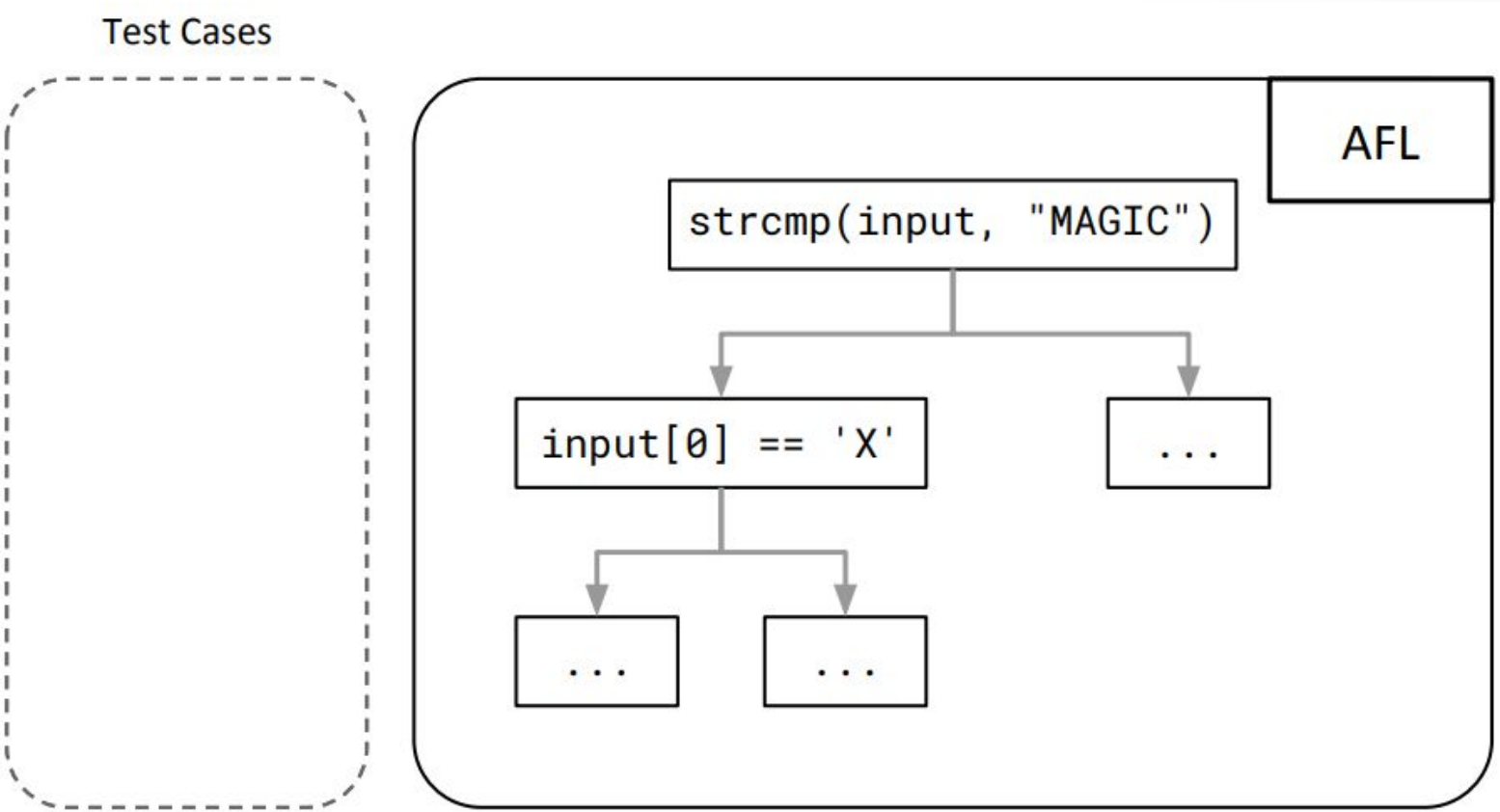
Design and Implementation

- ‘interesting’ inputs sharing
- launch SE when getting stuck

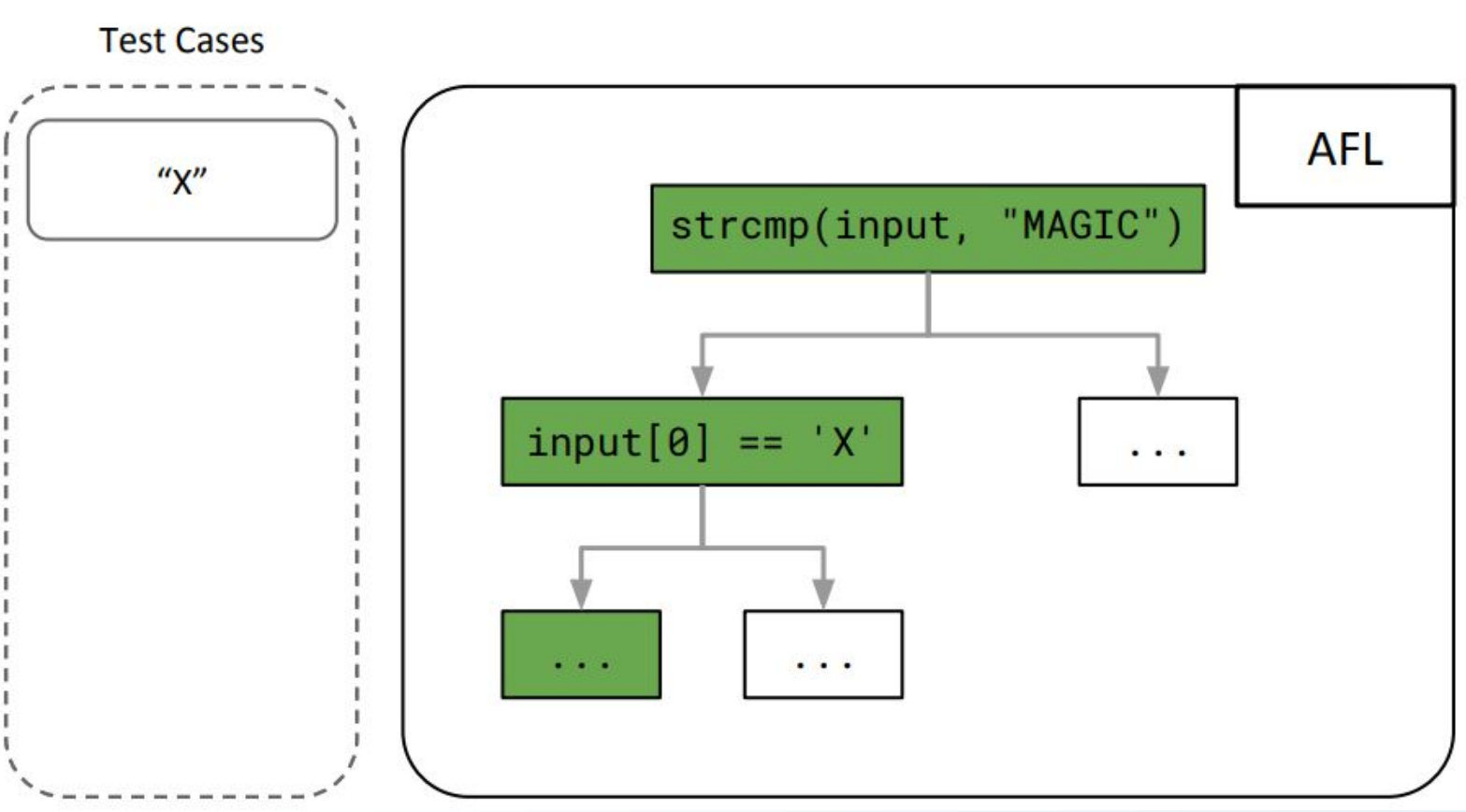


- generated inputs given back to fuzzer

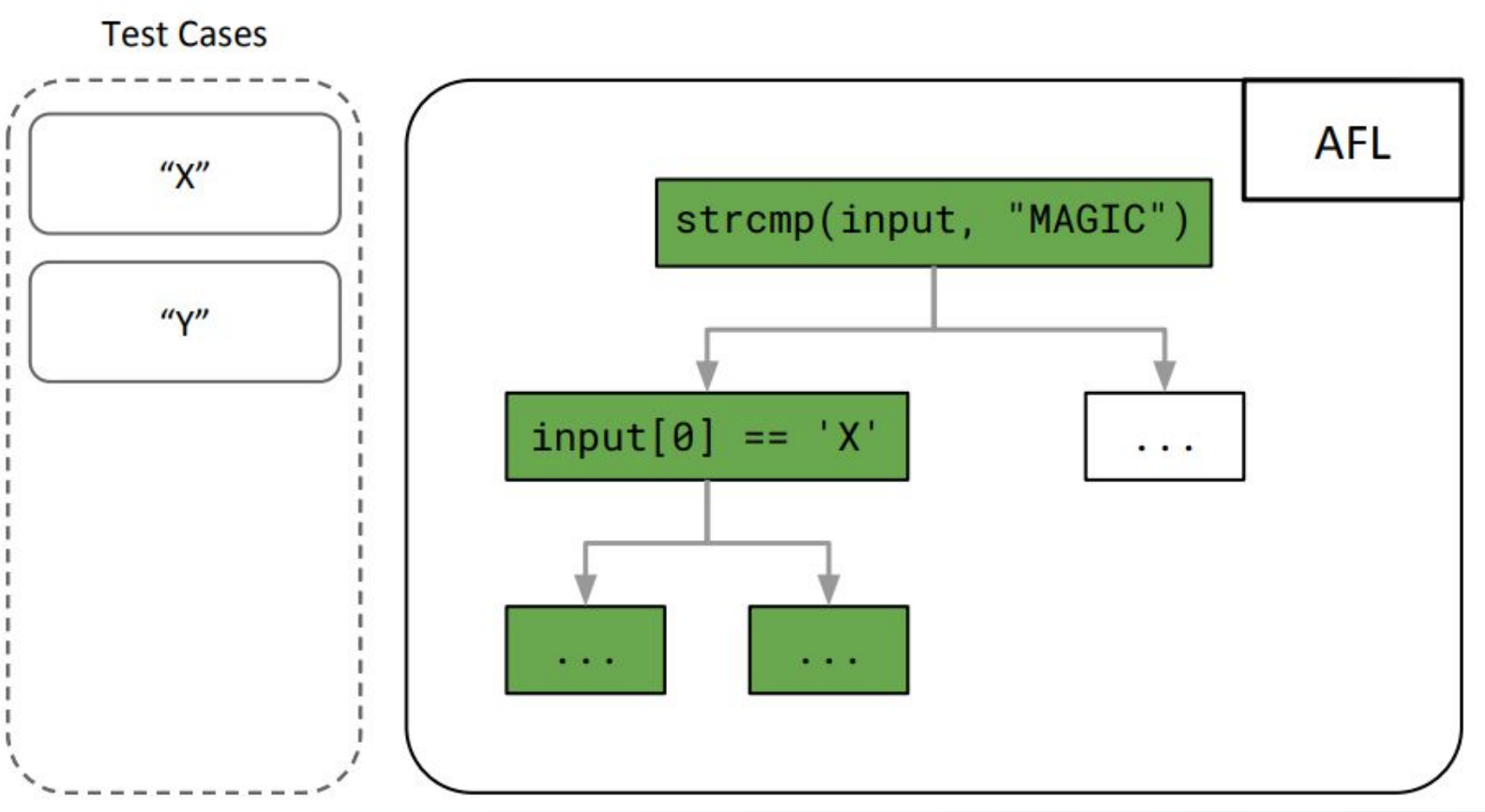
Hybrid Testing



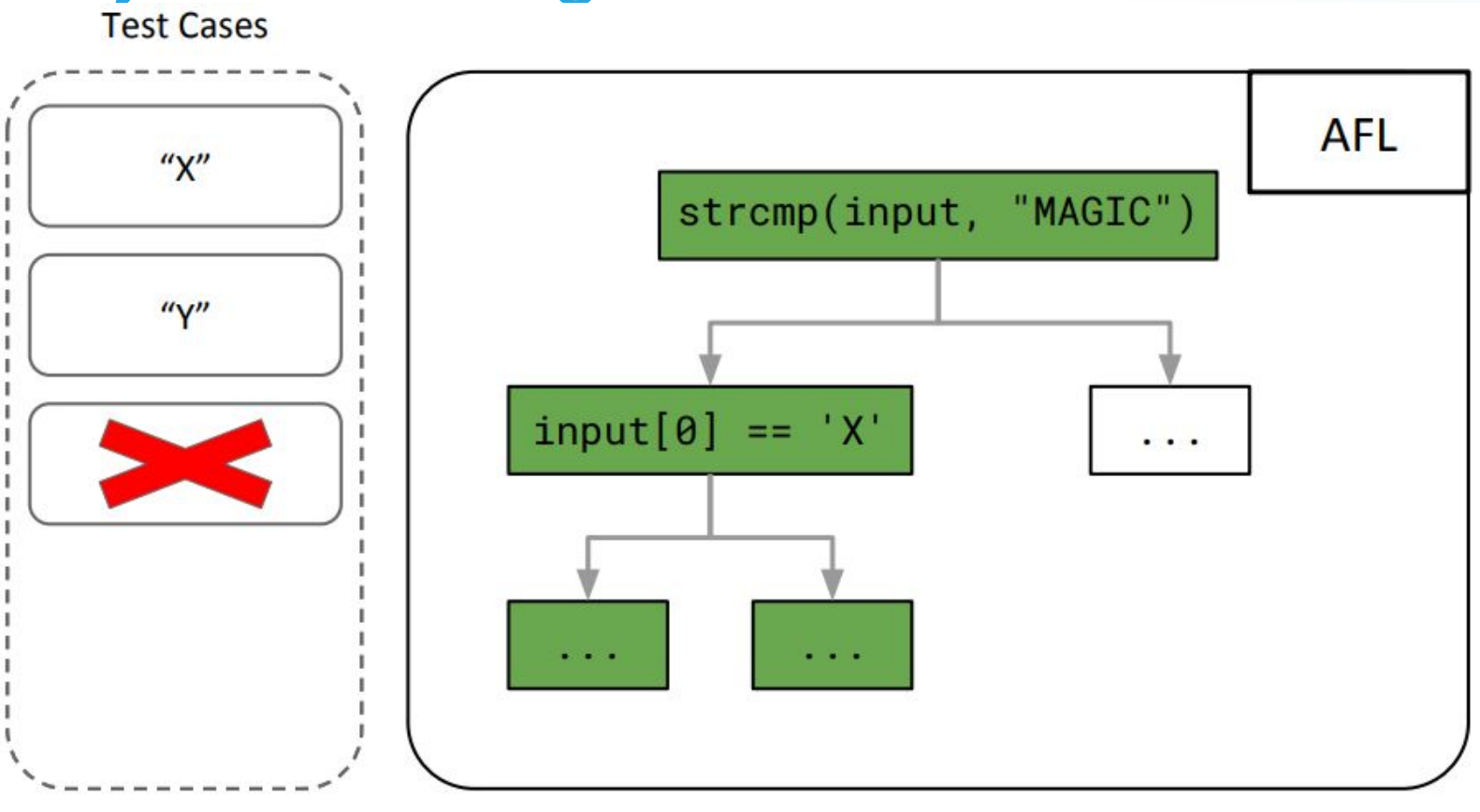
Hybrid Testing



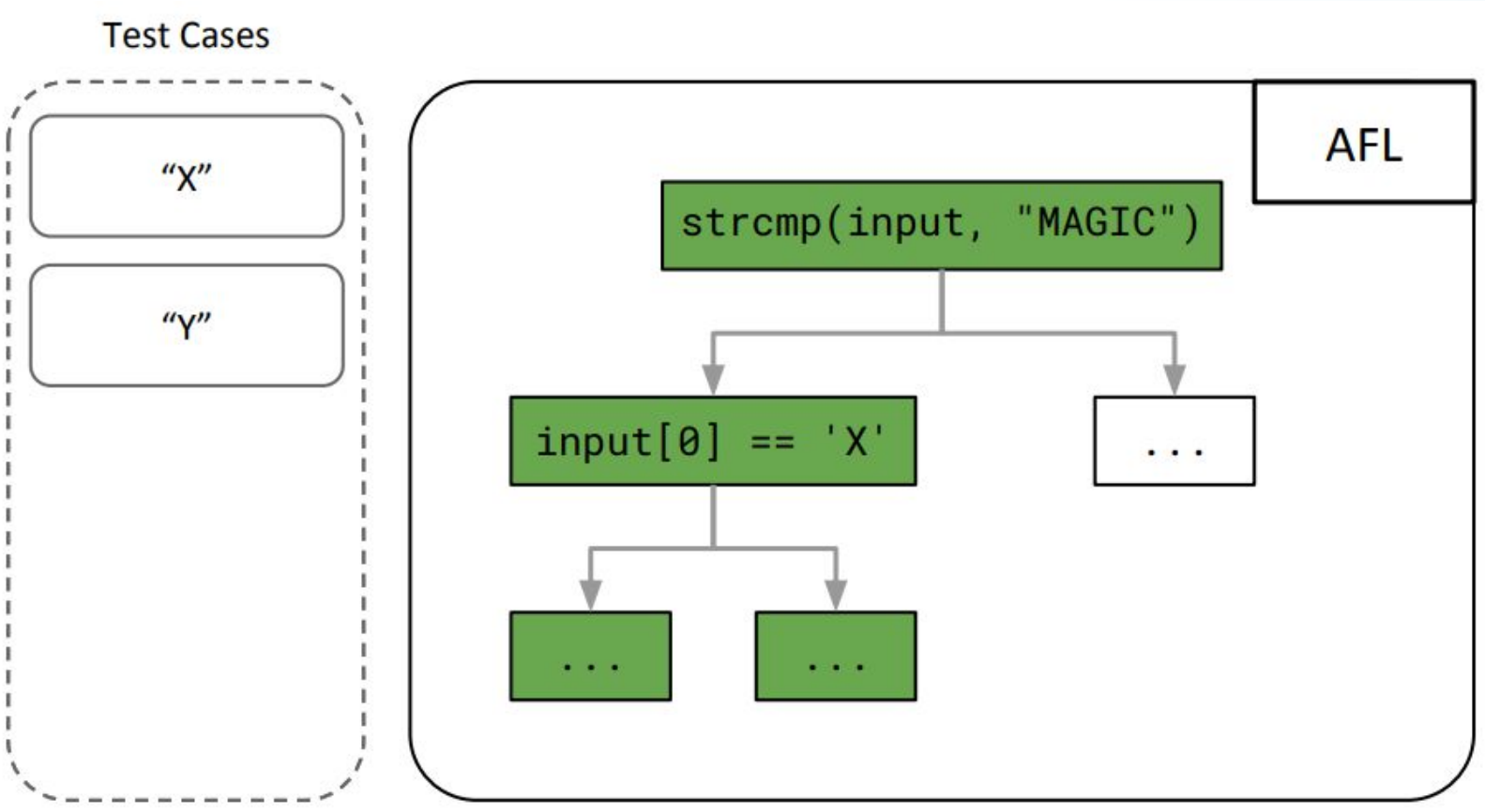
Hybrid Testing



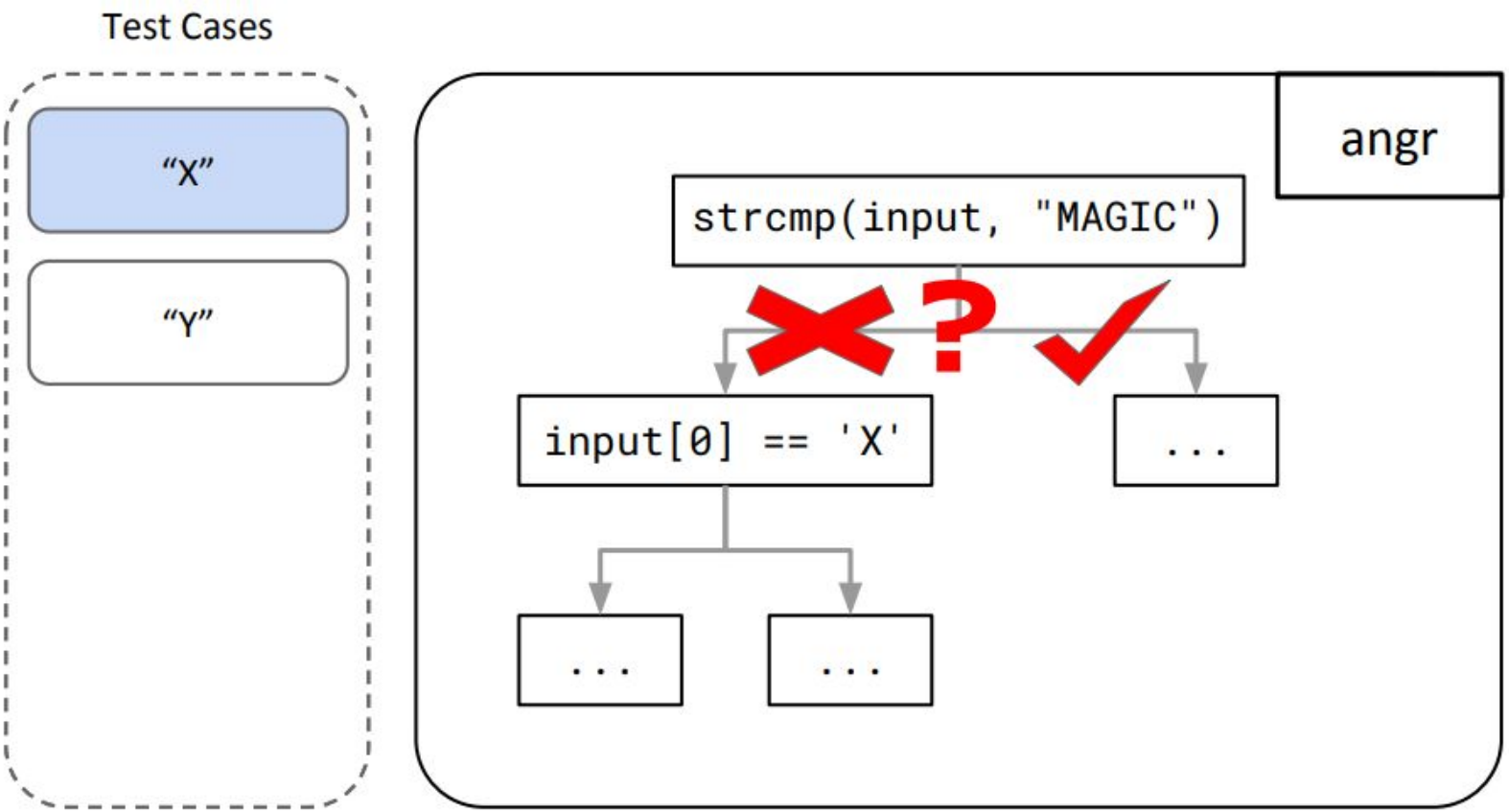
Hybrid Testing



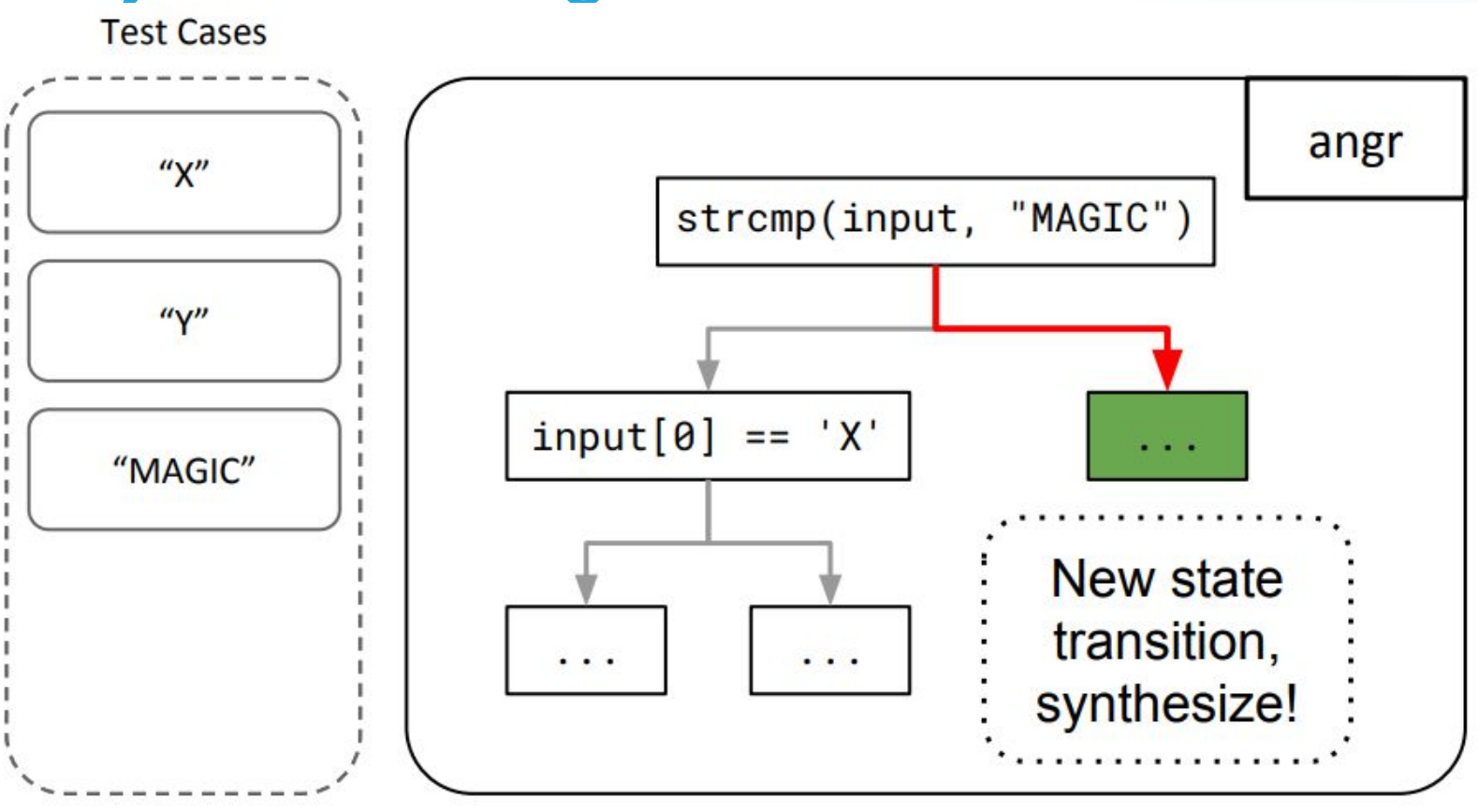
Hybrid Testing



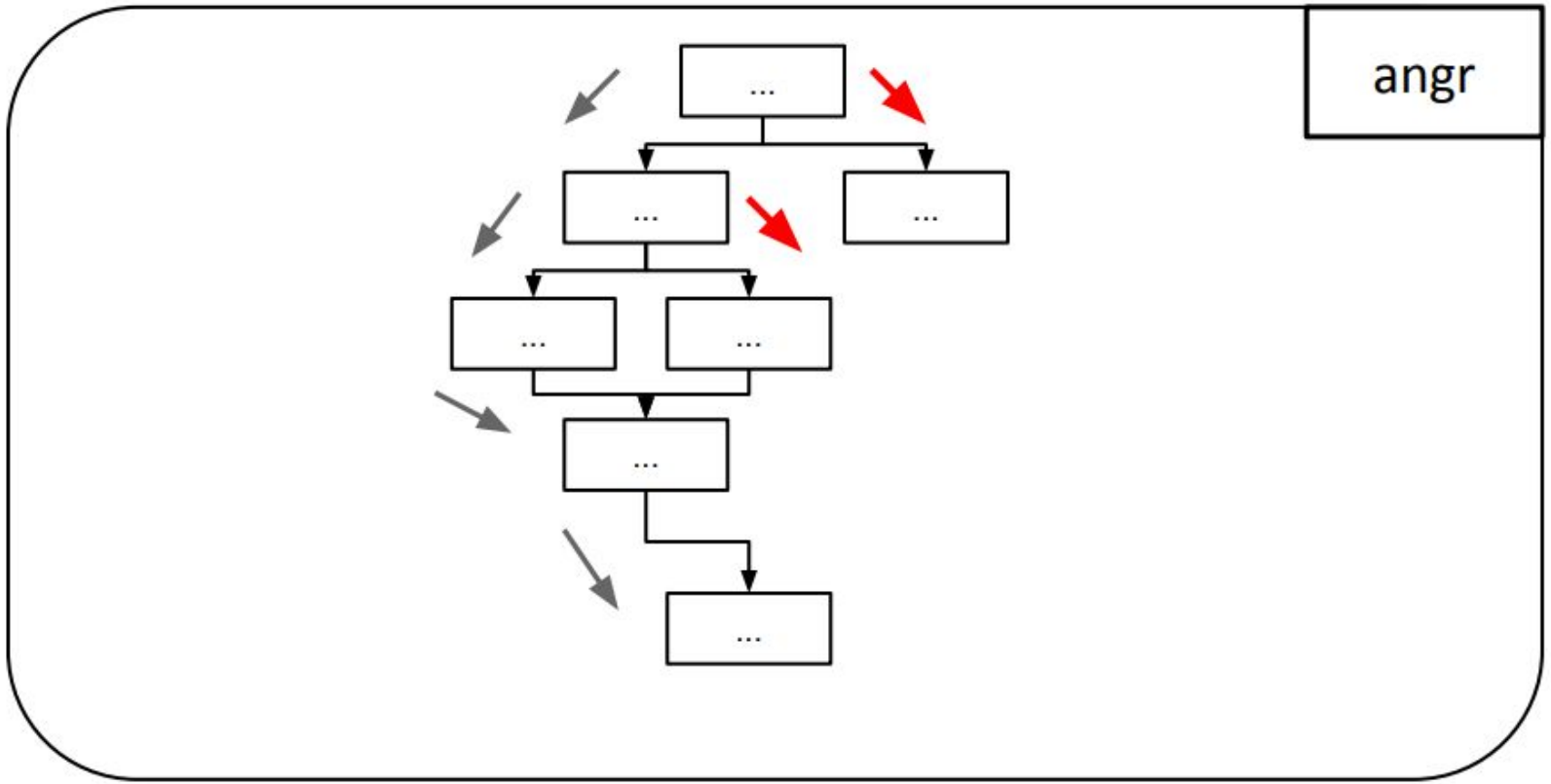
Hybrid Testing



Hybrid Testing

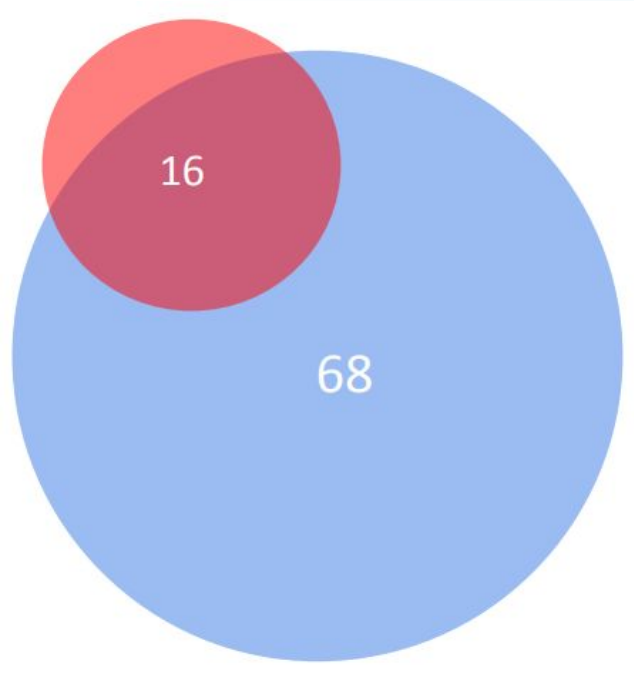


Hybrid Testing



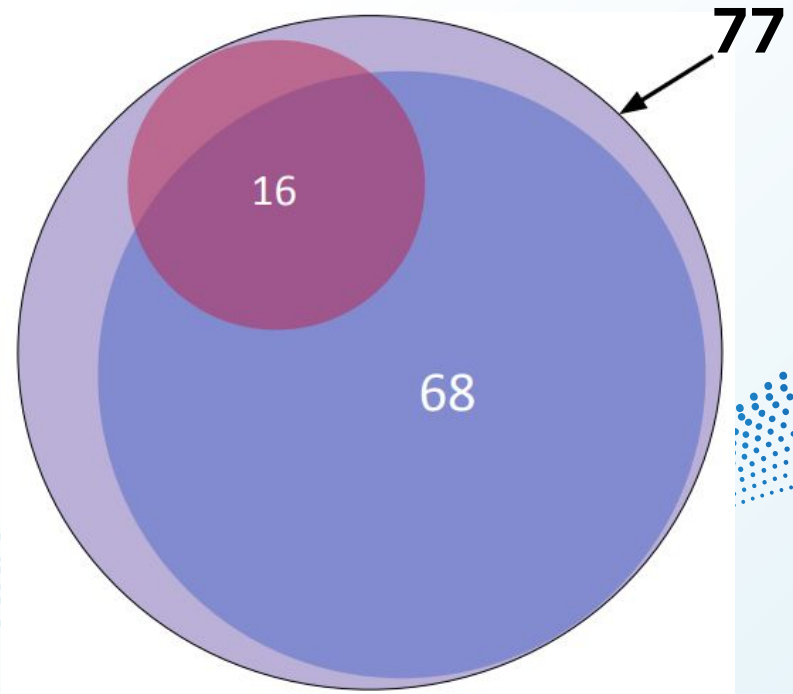
Evaluation

- Dataset
  - Darpa CGC binaries 128
- Symbolic Execution
  - crashed 16
- Fuzzing
  - crashed 68
- S & F shared
  - 13



Evaluation

- Driller
  - crashed 77/128 binaries
- Can cover all crashes



Send Hardest Problems My Way: Probabilistic Path Prioritization for Hybrid Fuzzing

Lei Zhao, Yue Duan, Heng Yin, Jifeng Xuan

NDSS 2019

Motivation

- State-of-the-art hybrid fuzzing approaches
  - **Demand launch**
    - Driller NDSS'16, Hybrid Concolic Testing ICSE'07
    - launch concolic execution when fuzzer gets stuck (blocked by condition checks
    - Assumptions
      - fuzzer in non-stuck state ⇒ concolic execution is not needed
      - stuck state ⇒ fuzzer cannot make progress
      - concolic execution is able to find and solve the hard-to-solve condition problems that block the fuzzer

Motivation

- State-of-the-art hybrid fuzzing approaches
  - **Optimal Strategy**
    - Markov Decision Processes with Costs ICSE'18
    - estimates the costs and always selects the best one
      - cost of fuzzing based on coverage statistics
      - cost of concolic execution based on constraints complexities
    - Assumptions
      - estimation is accurate and fast
      - decision making is lightweight enough

Motivation

- Systematic study is conducted to evaluate the strategies
  - **Demand launch**
    - the stuck state of a fuzzer is not a good indicator
    - not every missed branch requires concolic execution
    - cannot differentiate branches that block fuzzing from others
  - **Optimal strategy**
    - MDPC decision making is heavyweight
    - Throughput is significantly reduced
    - MDPC discovers fewer vulnerabilities

Probabilistic Path Prioritization

- Aim:
  - let concolic execution only solve the hardest problems
- Challenge:
  - how to **quantify** the difficulty of traversing a path for a fuzzer in a lightweight fashion
- Key idea:
  - Treat fuzzing as a sampling process
  - Estimate branch probabilities based on Monte-Carlo Method
  - Estimate path probabilities as Markov Chain of successive branches

Probabilistic Path Prioritization

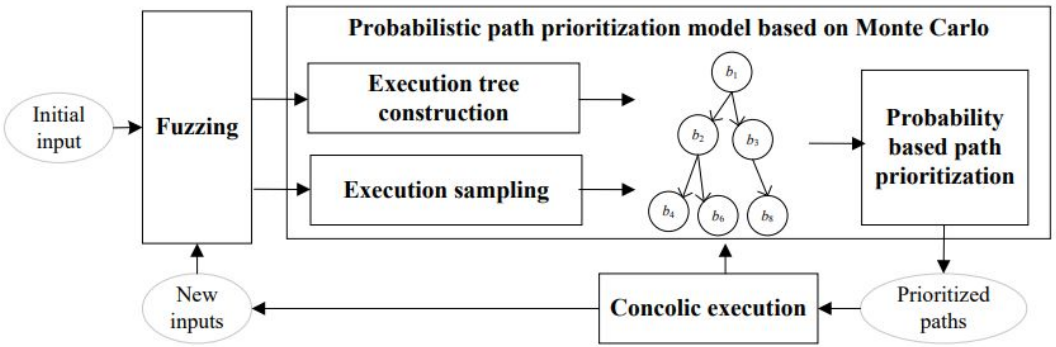
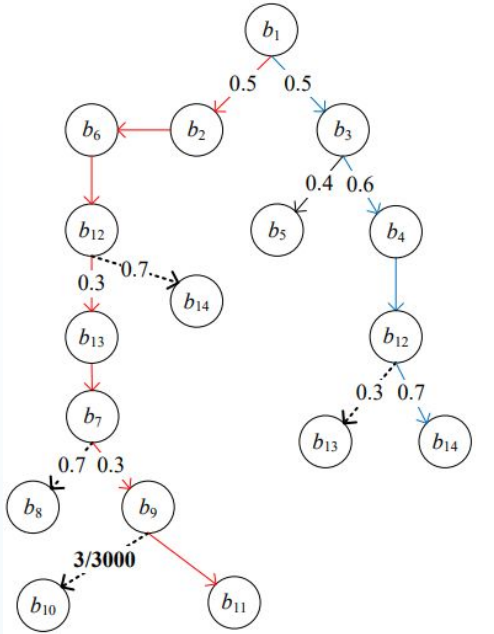


Fig. 3: Overview of DigFuzz



P_1=<b_1, b_2, b_6, b_{12}, b_{13}, b_7, b_9, b_{10}>, $P(P_1)$=0.000045
P_2=<b_1, b_2, b_6, b_{12}, b_{13}, b_7, b_8>, $P(P_2)$=0.063
P_3=<b_1, b_2, b_6, b_{12}, b_{14}>), $P(P_3)$=0.09
P_4=<b_1, b_3, b_4, b_{12}, b_{13}>, $P(P_4)$=0.105

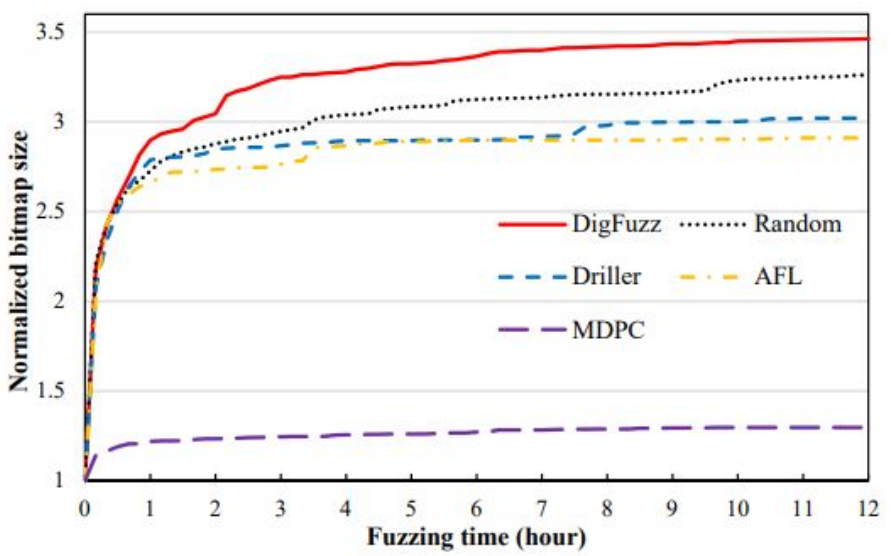
Fig. 5: The execution tree with probabilities

Evaluation

- Dataset
  - Darpa CGC binaries
    - 126 binaries
  - LAVA-M
    - 4 real-world binaries
- Baseline
  - AFL: pure fuzzing
  - MDPC: optimal strategy
  - Driller: demand launch
  - Random: concolic execution launched from the beginning (no path prioritization)

Evaluation

- Code coverage
  - DigFuzz, Random, Driller, and AFL are 3.46 times, 3.25 times, 3.02 times and 2.91 times larger than the base (code coverage of the initial inputs)



Evaluation

- Discovered vulnerabilities
  - Per Driller paper report, DigFuzz can achieve similarly with only half of the running time (12 hours vs. 24 hours) and much less hardware resources (2 fuzzing instances per binary vs. 4 fuzzing instances per binary)

TABLE II: Number of discovered vulnerabilities

	= 3	≥ 2	≥ 1
DigFuzz	73	77	81
Random	68	73	77
Driller	67	71	75
AFL	68	70	73
MDPC	29	29	31

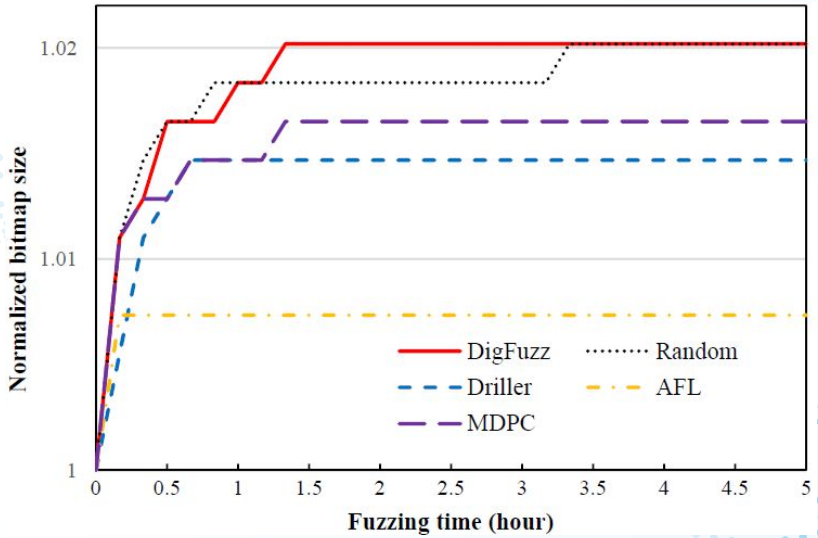
Evaluation

- Contribution of concolic execution
  - More binaries aided by concolic execution (Aid.) ⇒ CE launched in more binaries
  - More imported and derived inputs from concolic execution (Imp. and Der.) ⇒ better quality for generated inputs
  - More crashes are triggered by inputs from concolic execution ⇒ more effective in finding vulnerabilities

	Ink.	CE	Aid.	Imp.	Der.	Vul.
DigFuzz	64	1251	37	719	9,228	12
	64	668	39	551	7,549	11
	63	1110	41	646	6,941	9
Random	68	1519	32	417	5,463	8
	65	1235	23	538	5,297	6
	64	1759	21	504	6,806	4
Driller	48	1551	13	51	1,679	5
	49	1709	12	153	2,375	4
	51	877	13	95	1,905	4

Evaluation

- LAVA-M consists 4 small applications
  - DigFuzz achieved better code coverage
  - Random catched up because the programs are small



Thank you!

Questions?