Arvin: Greybox Fuzzing Using Approximate Dynamic CFG Analysis

Sirus Shahini, Mu Zhang, Mathias Payer, Robert Ricci

THE UNIVERSITY OF UTAH

EPFL

Introduction: Fuzz Testing

- Fuzz testing: The most popular method to discover bugs
- Running programs with random inputs, looking for crashes
- Thousands of **security critical** bugs in the past few years
- The core challenge:
  - We must find the right input(s) …
  - There are infinite number of inputs…

Arvin's Contributions

- A new way of **instrumenting** programs under test
- Better **context** for analysis of dynamic program behavior
- Better program **coverage, faster**
- Better **bug finding**: 50 bugs in 17 programs

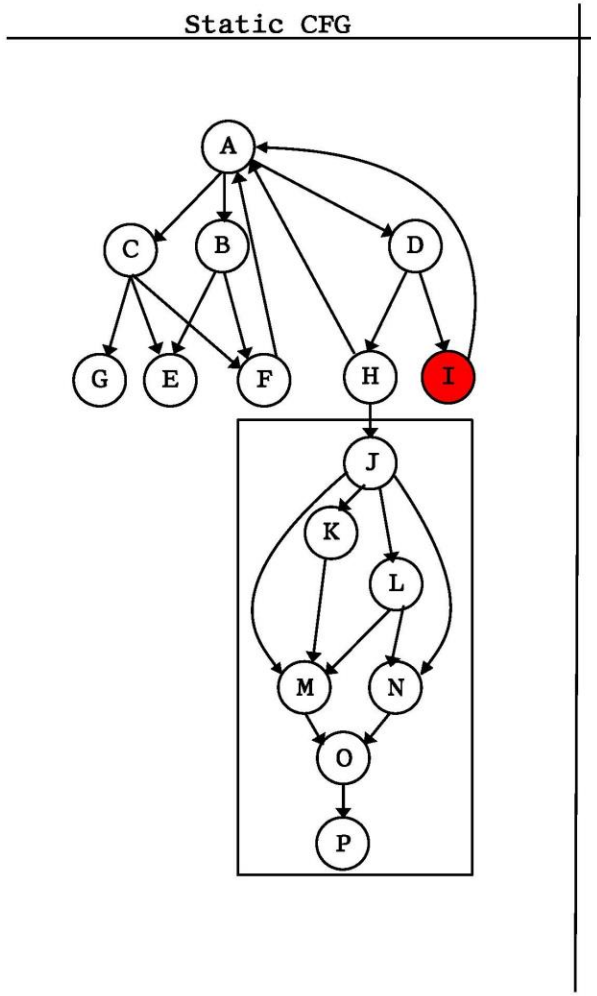
Background

- Time is GOLD
- High-level goal: good coverage and many crashes
  - Usually measured in basic blocks or edges
  - Grey-box fuzzing: limited visibility into program under test
- Better inputs : better coverage
- Prioritization makes a big difference
- Understanding program behavior requires **context awareness**

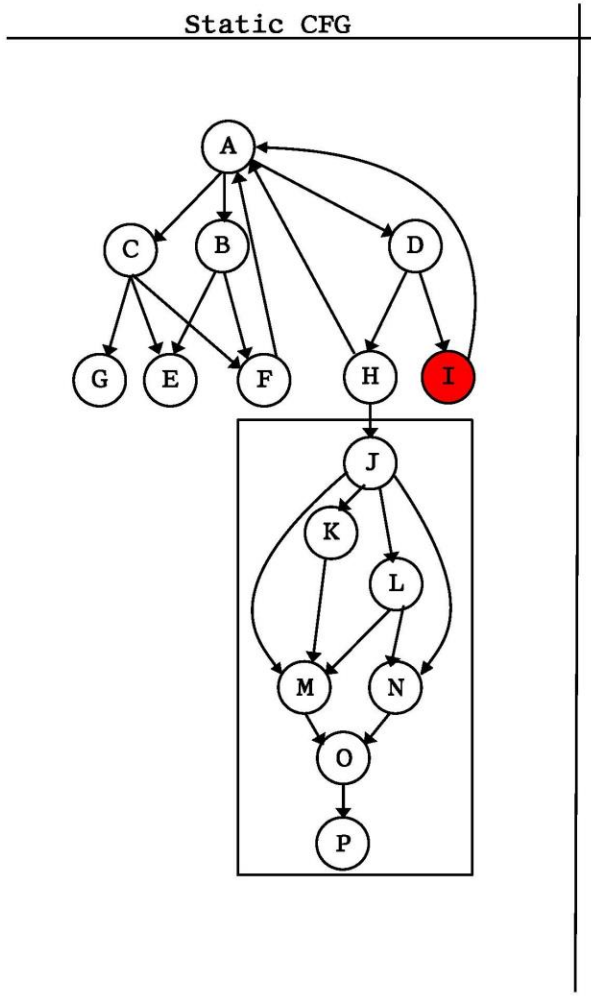
Arvin: Context-Aware Fuzzing

- Understanding the PUT's context using control-flow graphs
- Dynamic control-flow graphs at basic block level
- Most inputs end very close to the PUT's entry point
  - Getting farther from entry and closer to more interesting areas
- Prioritize inputs that are *high quality*
  - First, get high coverage
  - Then, heavily exercise those inputs

Example



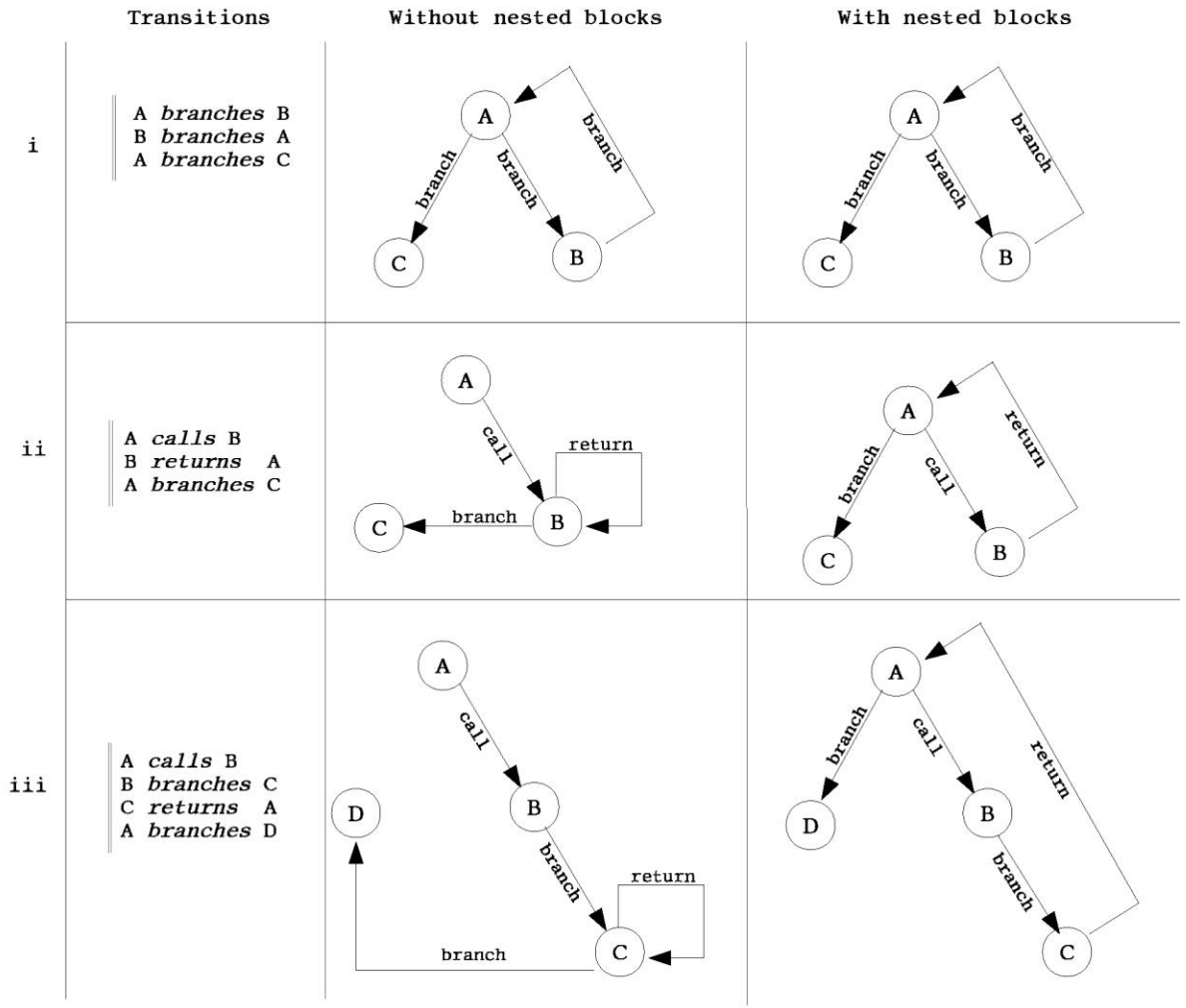
Example



Building Dynamic Graphs

- Graphs are complex – most fuzzers use only bitmaps
- Building DCFGs at runtime is challenging for a greybox fuzzer
- Arvin's core instrumentation: The DCFG runtime library
  - Instruments PUT in-memory
- Independent and nested basic blocks
  - Treat `call` instructions differently from most fuzzers
  - For Arvin, they start a new *nested* basic block

Building Dynamic Graphs: Nested Blocks



Calculating Priorities From DCFGs

- New coverage
  - Not just new blocks, but new edges between them
- Depth
  - Get far from PUT entry point
- Target specific basic blocks
  - Representing potentially vulnerable functions
  - … and specific paths to reach them

Balancing the Priorities

A) Input queue without prioritization*

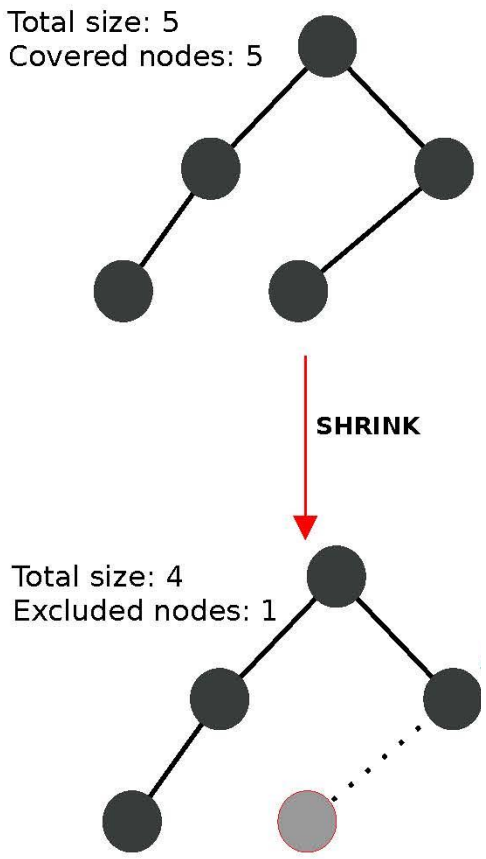
Offsetting the Analysis Load

- Millions of executions for a single bug
- We need to reduce the cost of graph analysis
- Some nodes are not needed in later iterations
- Arvin makes use of *directed approximation*
  - Strategy: Remove instrumentation for blocks that run the most
  - Reduces instrumentation and analysis cost
  - They were generally not giving much information anyway
  - Remember them between runs of the PUT

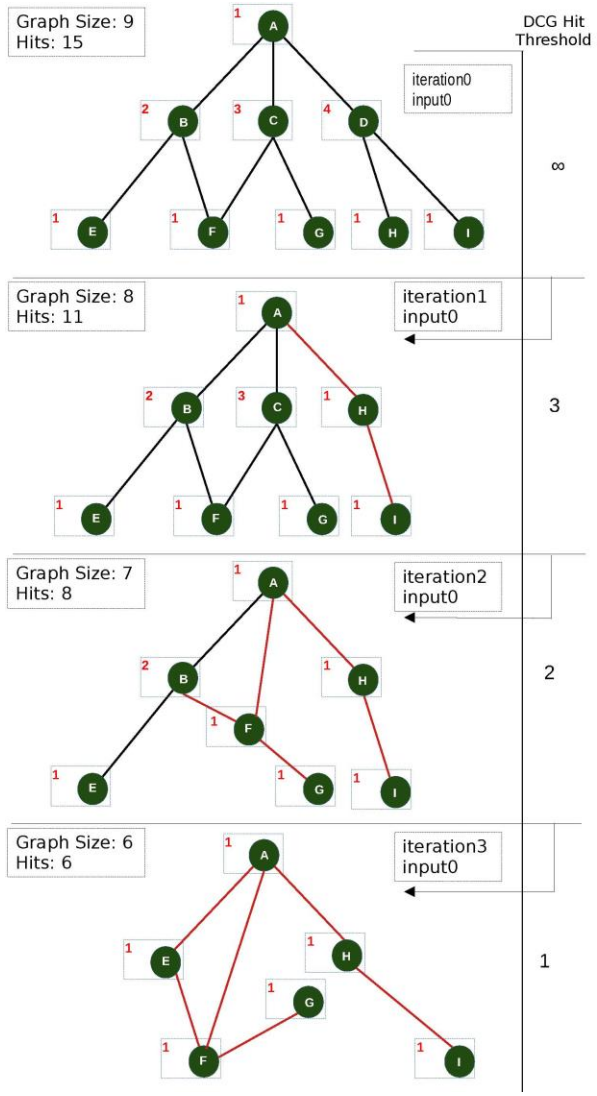
DCFG Approximation

- Arvin's shrink-grow cycle: Decremental CFG Growth (DCG)
- Hit counts matter: DHT
- Shrink: Choose candidate nodes for exclusion
- Grow: Discover and add new nodes
- Save time to invest on more efficient coverage growth

Example: Shrink and Grow

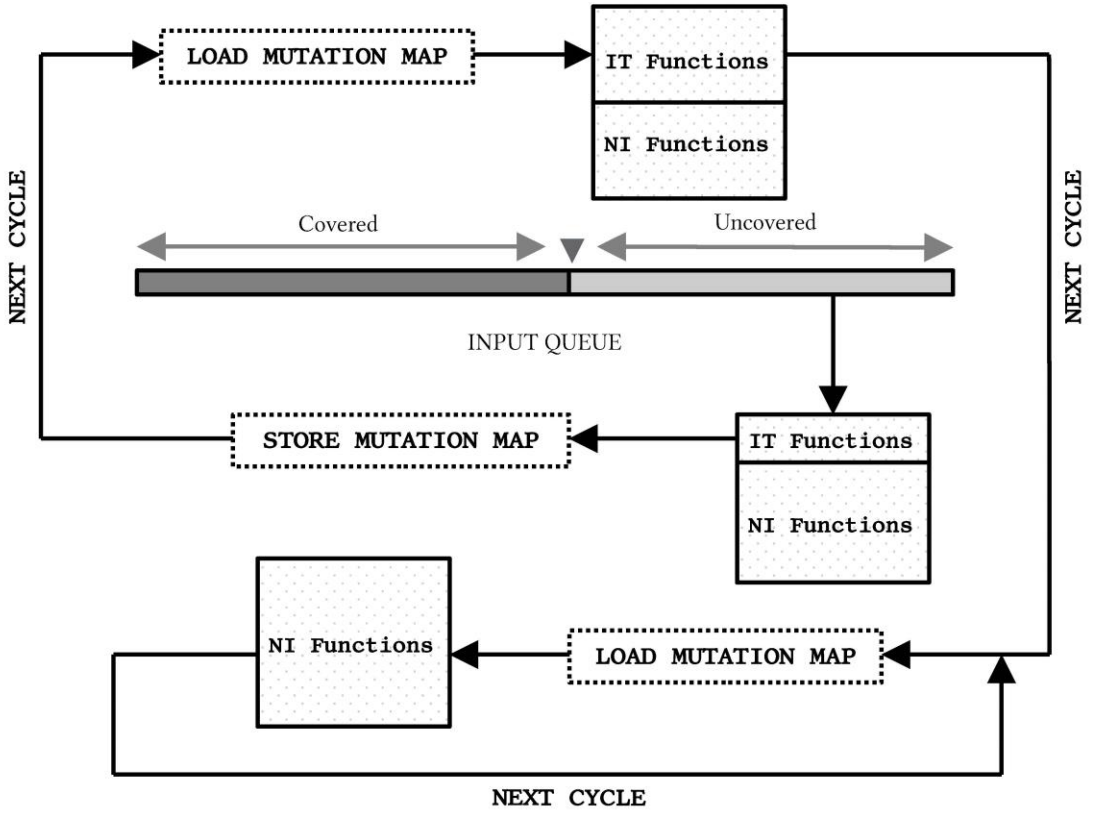


Varying Thresholds

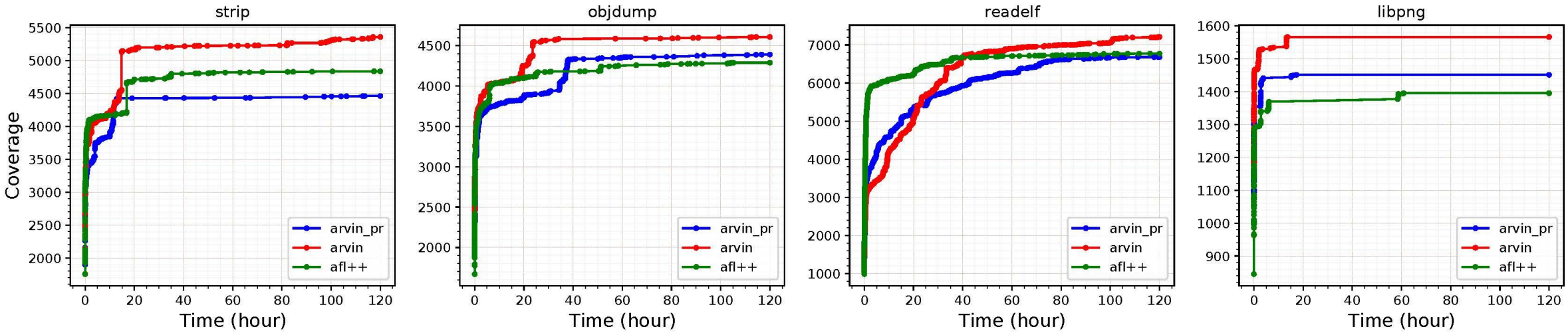


Adaptive Mutation

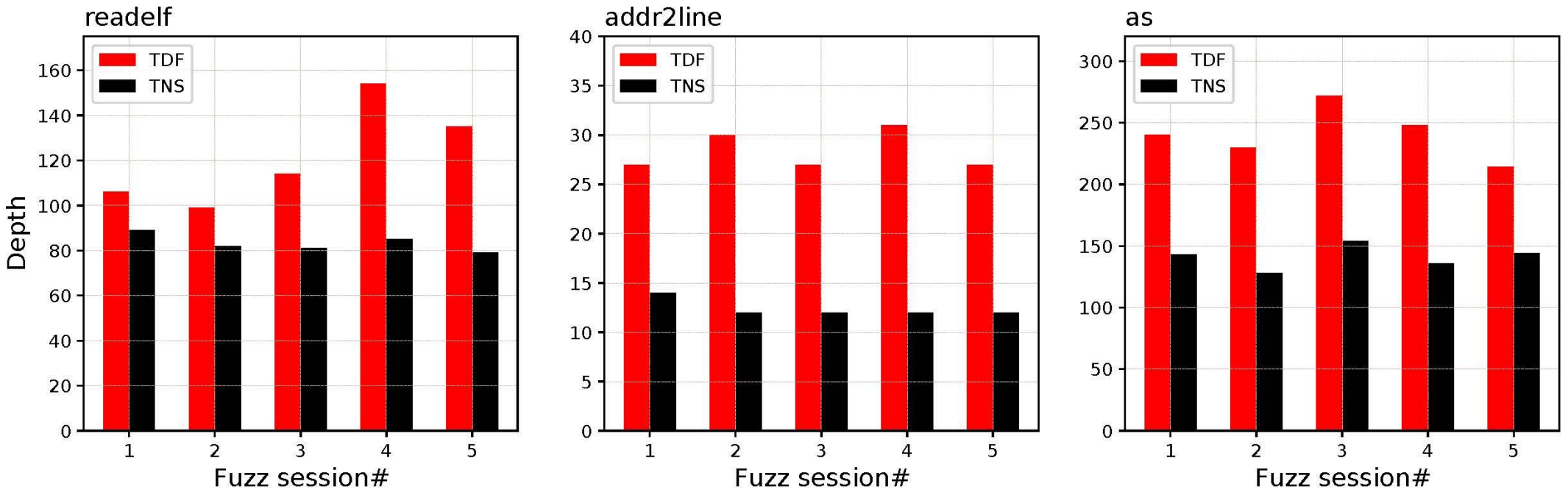
- Deterministic and Non-deterministic mutations
- Deterministic (IT)
  - Pick a good input
  - Try all of our mutations on it
  - Tends to find crashes
- Random (NI)
  - Explore inputs with random muts.
  - Tends to expand coverage
- Adjust rates over time



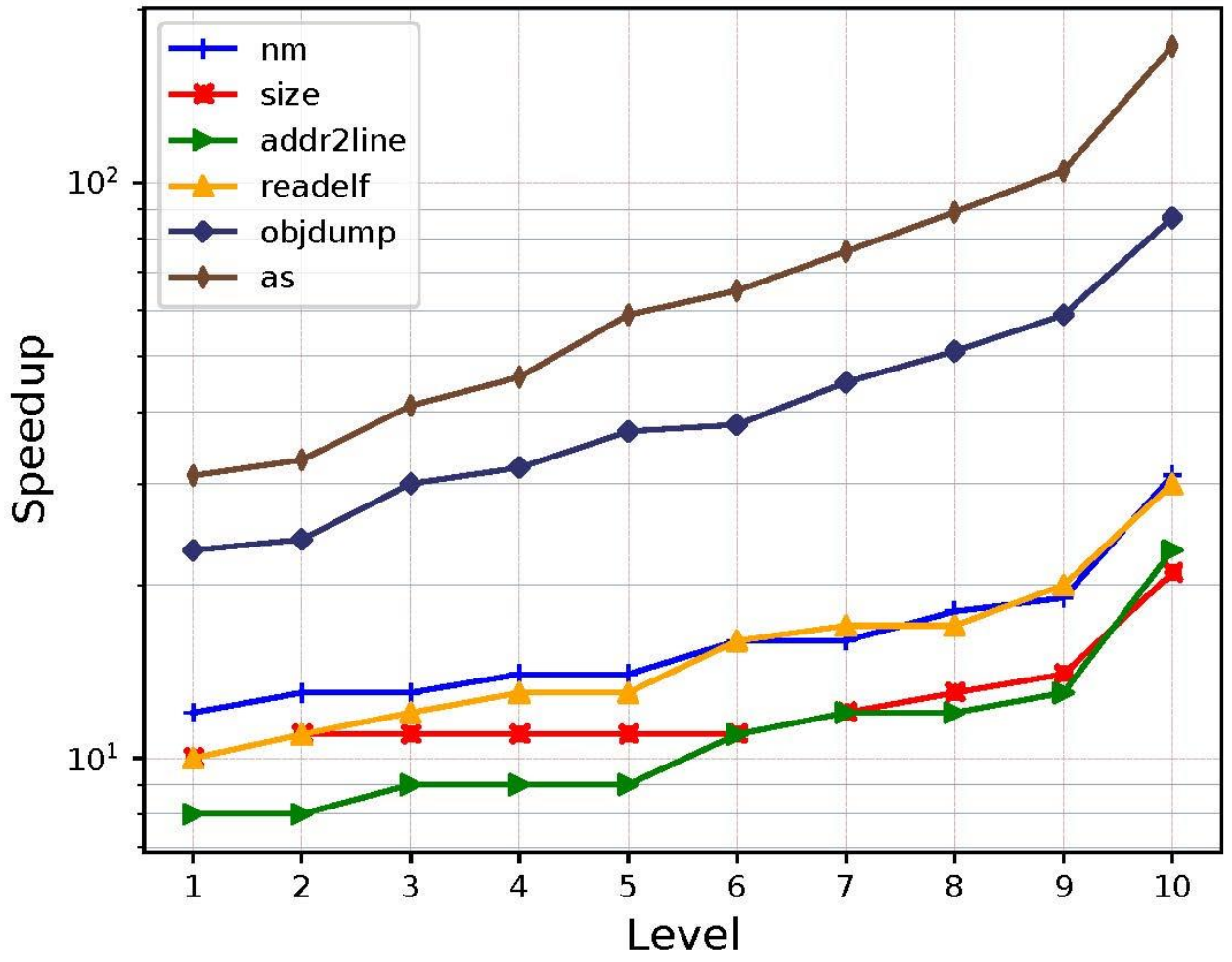
Evaluation: Testing coverage growth



Evaluation: Depth



Evaluation: Approximation



Discovered bugs

Program	Arvin	AFL	hf	hf-qemu	APP	APP-qemu	TortoiseFuzz	Angora*	VUzzer	ParmeSan
GNU assembler (v 2.35, 2.36, 2.37)	13	0	4	1	5	1	3	0	0	0
unzip (v 6.00)	4	2	2	0	3	1	0	0	0	0
gif2png (v 2.5.8-1)	2	2	2	2	2	2	2	2	0	2
readelf (v 2.35)	1	0	0	0	0	0	0	0	0	0
bison (v 3.5, 3.7)	7	3	4	2	4	4	3	1	0	1
fig2dev (v 3.2.8a)	3	1	2	1	2	2	2	1	0	0
flvmeta (v 1.2.2)	1	0	1	0	1	0	0	0	0	0
nasm (v 2.15.05)	1	0	0	0	0	0	0	0	0	0
ibmtpm (last commit May 15 2021)	1	0	0	0	0	0	0	0	0	0
libraw (v 0.21.0)	2	0	0	0	1	0	0	0	0	0
pnmtopng (v 1.17.dfsg-4)	2	1	1	1	2	1	1	0	0	0
pnmtojpeg (v 1.17.dfsg-4)	2	0	2	2	1	1	1	0	0	0
pnmtofiasco (v 1.17.dfsg-4)	2	2	2	2	2	2	2	2	0	2
pstopnm (v 1.17.dfsg-4)	3	0	2	1	1	1	1	0	0	0
psnup (v 1.17.dfsg-4)	2	0	1	1	2	1	1	0	0	0
xpdf (v 4.03)	1	0	0	0	0	0	0	0	0	0
xvid (last version from http://svn.xvid.org/trunk)	3	0	0	0	1	0	0	0	0	0
Total	50	11	23	13	27	16	16	6	0	5

Related Work

- angr framework [Shoshitaishvili et al, IEEE S&P '16]
  - We use it to find basic blocks, etc.
- Inspiration
  - AFL [Zalewski], AFL++ [Fioraldi et al, WOOT '20], and others
- Fuzzing with CFGs
  - ParmeSan [Osterlund et al, USENIX Security '22]
- Fuzzing with adaptive instrumentation
  - Full Speed Fuzzing [Nagy et al, IEEE S&P '19]

More In The Paper

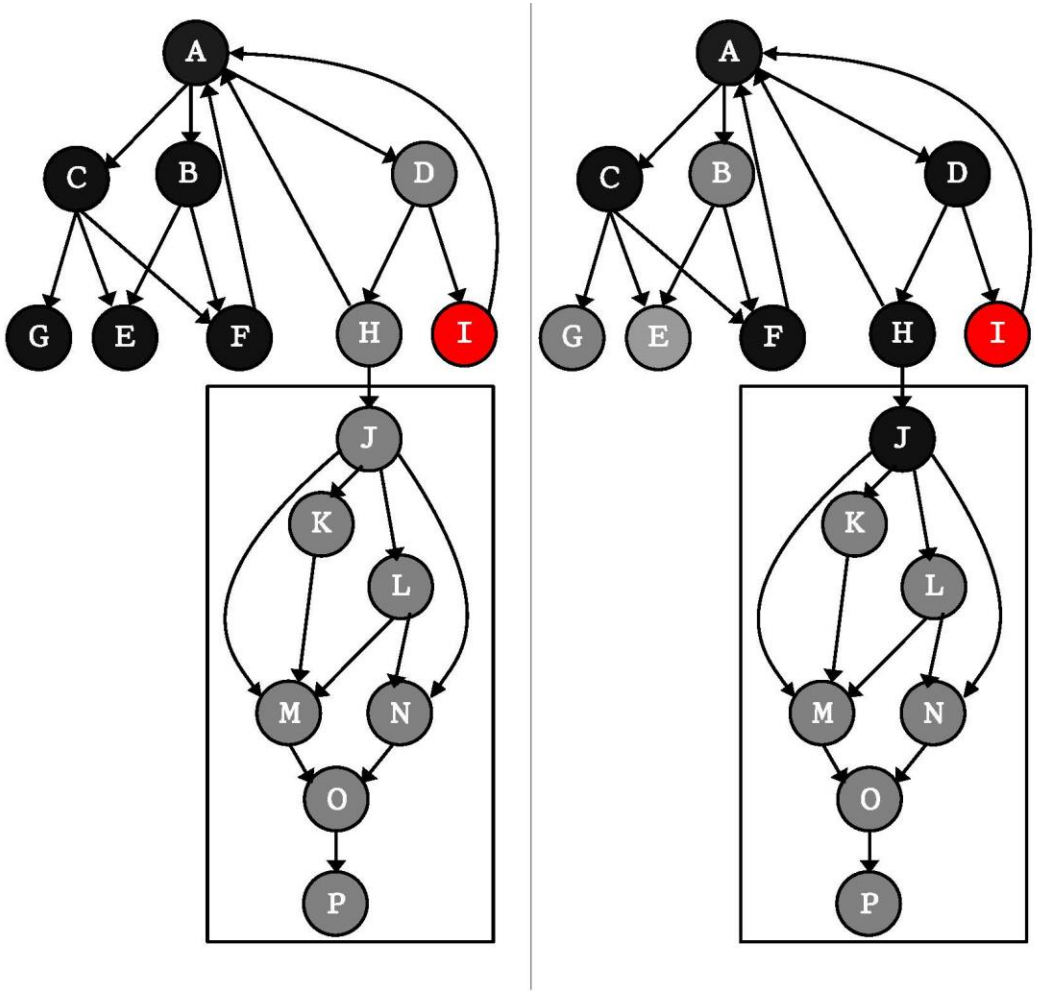
- Parallel fuzzing
- Detailed examples
- More evaluation

Try Arvin

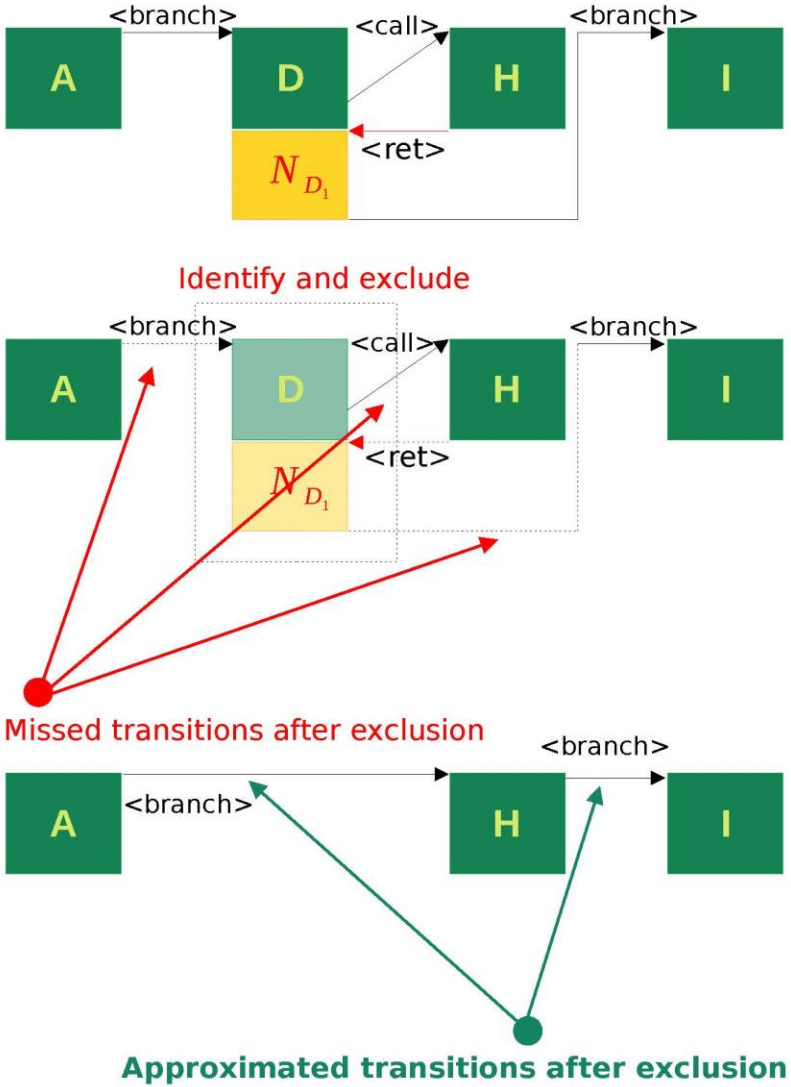
- Contact: sirus.shahini@gmail.com
- Open source: https://github.com/0xsirus/arvin

Questions?

Example

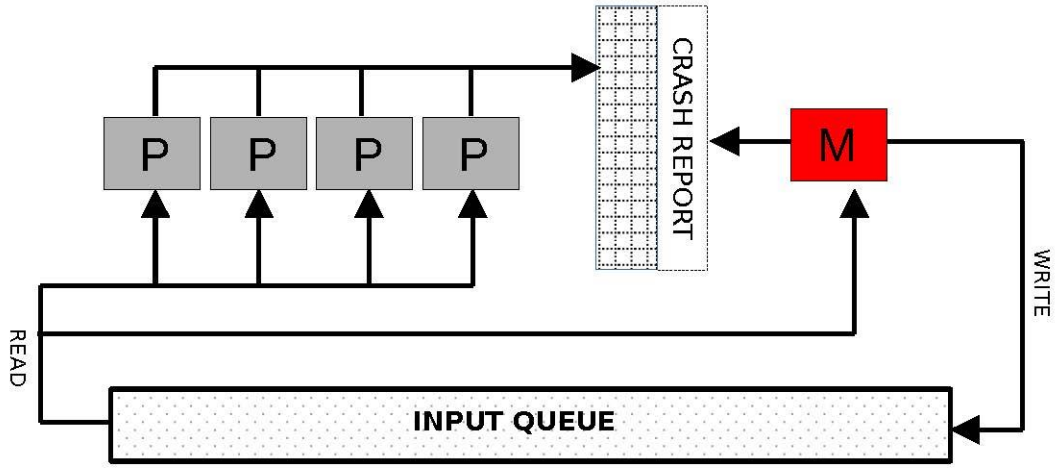


Exclusion in Detail

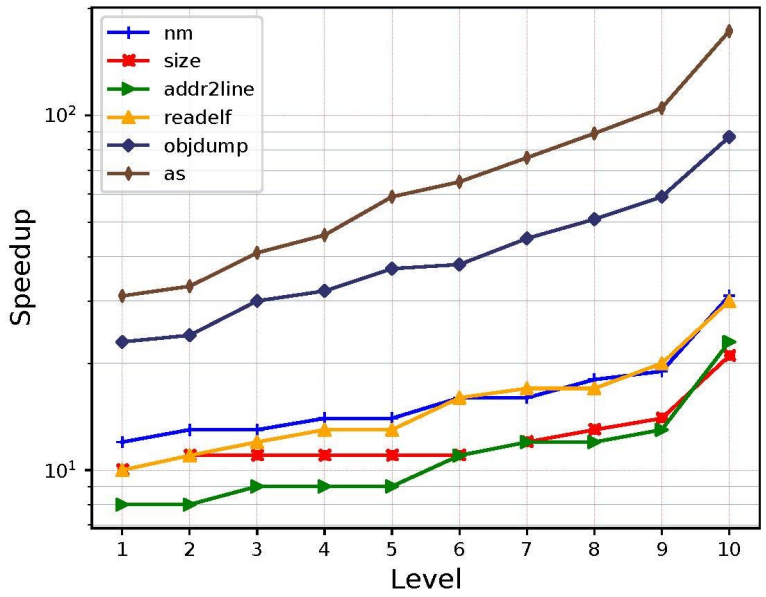


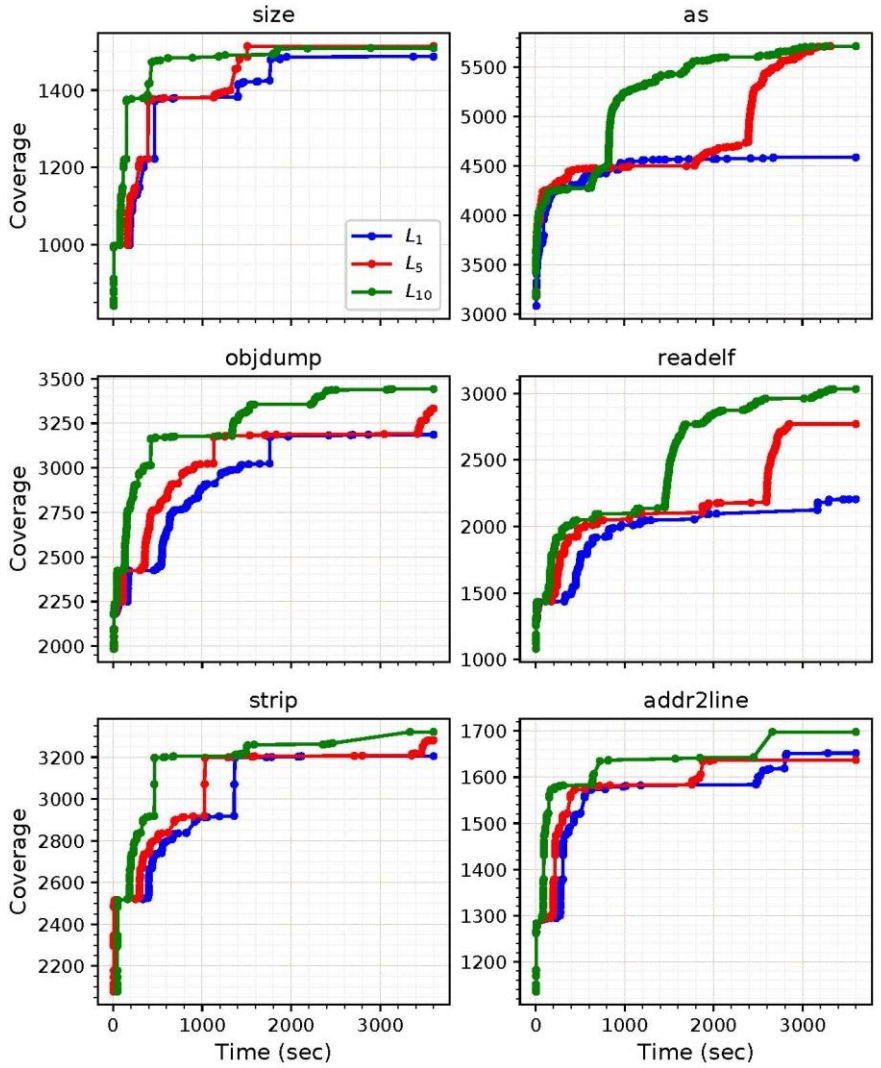
Parallel Fuzzing

- Separate instrumentation libraries for different fuzzing modes
- Cooperative parallel fuzzing



Evaluation: Approximation





Evaluation: Parallel fuzzing