QSYM : A PRACTICAL CONCOLIC EXECUTION ENGINE TAILORED FOR HYBRID FUZZING

Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang and Taesoo Kim,

FINDING SECURITY BUGS

- Fuzzing
  - Automated test to monitor exceptions (crashes & memory leaks)

- Pro: general inputs (loose branch condition: x<1000)
- Con: specific inputs

FINDING SECURITY BUGS

- Concolic Execution - concrete execution drive the symbolic execution through specific path
  - Symbolic Execution
    - Execution through all paths
  - Concrete Execution
    - Executing with values

- Pro: specific inputs (narrow conditions: x == 0xfdsgs)
- Con: path explosion - feasible paths in a program grows exponentially with an increase in program size

FINDING SECURITY BUGS – CONCOLIC TESTING

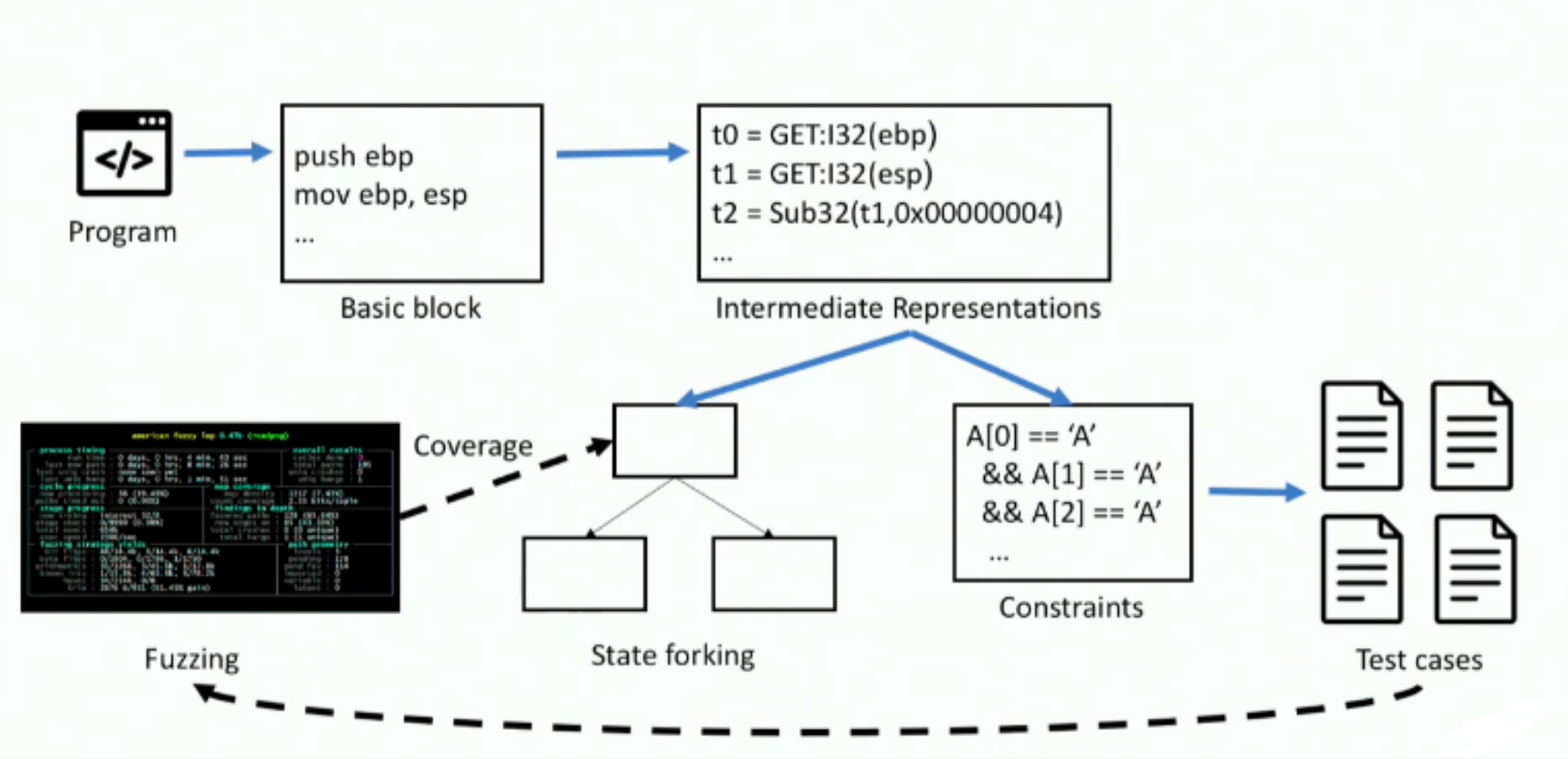
```
y = read();
 z = y * 2;
if (z == 12) {
        fail();
   } else {
        printf(“ - ");
}
```

- Read in 5 (concrete execution)
- Constraints
  - $\lambda * 2 == 12$
  - $\lambda * 2 != 12$
- Termination results in a concrete value (test cases)

HYBRID FUZZING

- Combination of techniques
  - Fuzzing – explore trivial input spaces
  - Concolic – solve complex branches
- Forking when needed
- Proven to work by Driller
  - 6 new crashing inputs not found by using individually

HYBRID FUZZING



HYBRID FUZZING - PROBLEMS

- Slow to generate constraints
- No support for complete system calls
- Bad at generating test cases

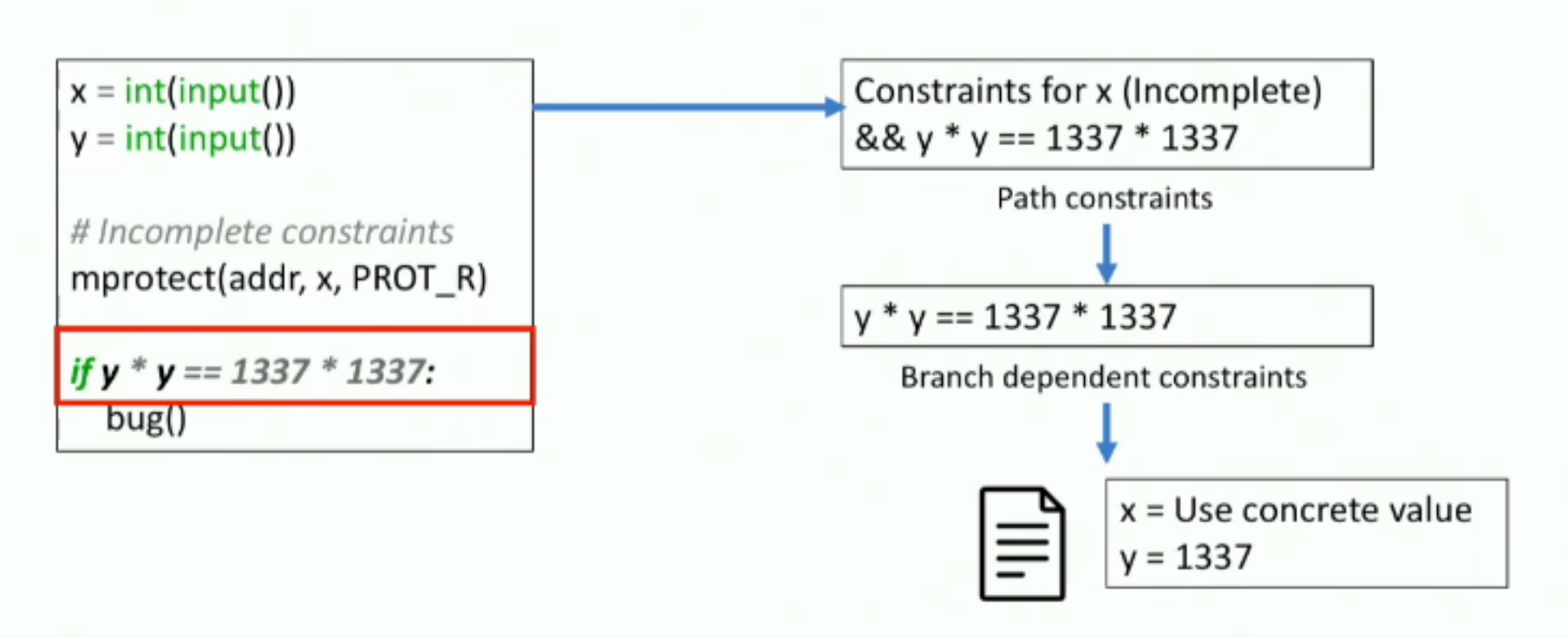
QYSM

- Remove IR translation layer to reduce overhead (minimal symbolic emulation)
- Concrete execution to model external environment – support to system calls (models minimal system calls)

  mprotect(addr, sym_size,PROT_R)

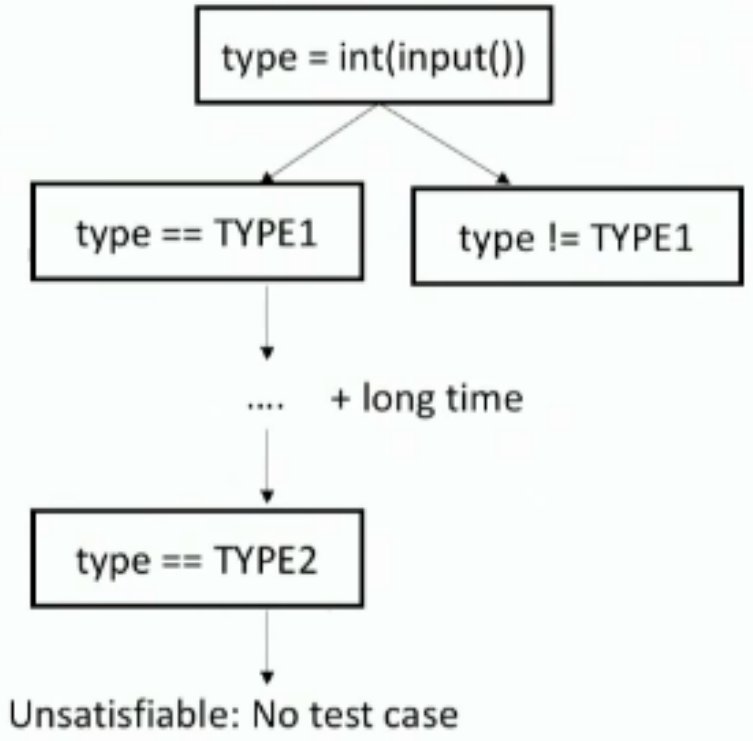
  mprotect(addr, conc_size,PROT_R)
- Smart constraint solving
  - Incomplete constraints (efficiency) – Unrelated concrete elimination
    - Only solve constraint associated to branch
  - Overly constrained path (solve portion)

QSYM - INCOMPLETE CONSTRAINTS

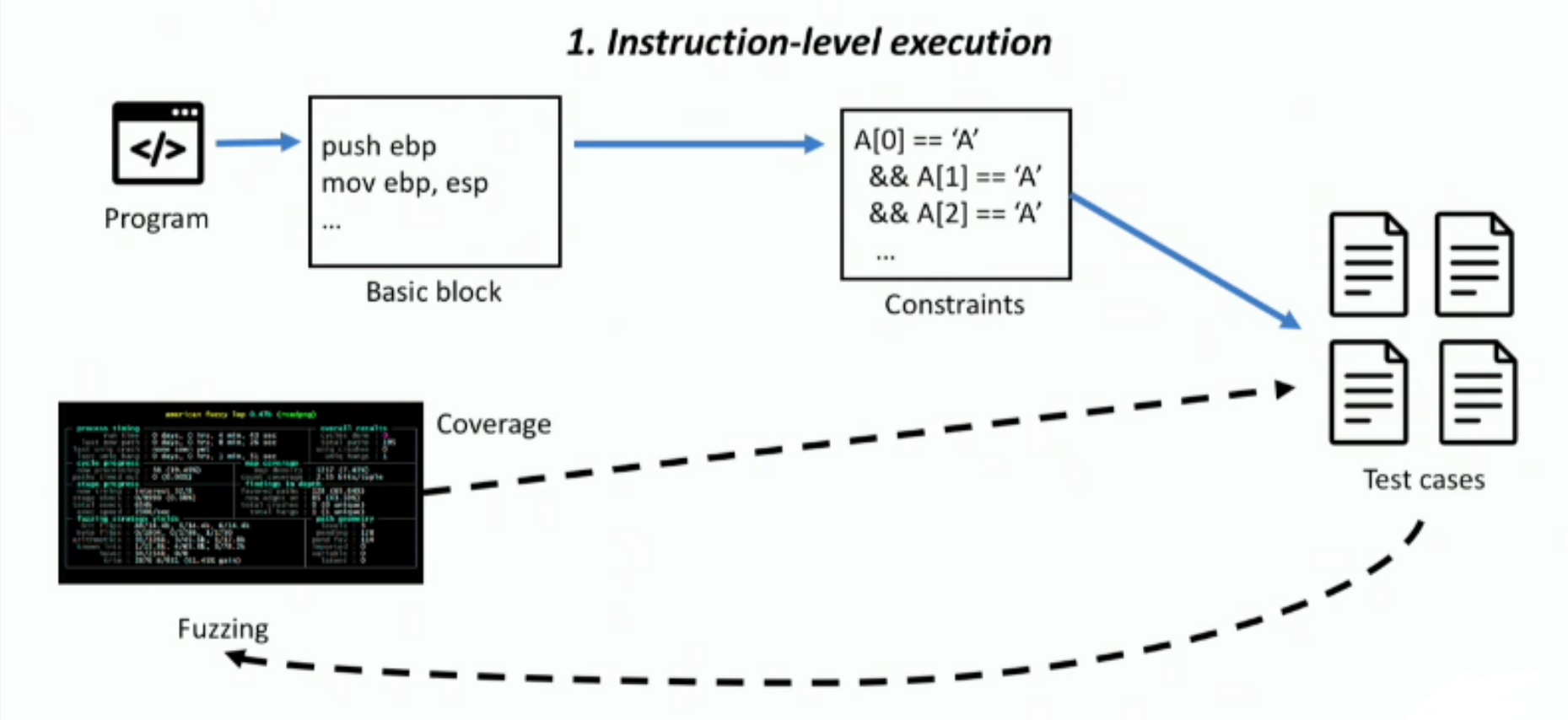


QSYM – OVERLY CONSTRAINED PATHS

```
type = int(input())

if type == TYPE1:
  parse_TYPE1()

...

if type == TYPE2:
   parse_TYPE2()
```



QSYM


1. Instruction-level execution
Program
push ebp
mov ebp, esp
...
Basic block
A[0] == 'A'
&& A[1] == 'A'
&& A[2] == 'A'
...
Constraints
Test cases
Coverage
Fuzzing

QSYM – BASIC BLOCK PRUNING

- Detect repetitive basic blocks and prunes them for symbolic execution with subset of constraints
- Counts frequency of basic blocks and at runtime selects the repetitive blocks to prune
- If a basic block is executed frequently then it will stop generating constraints for it
- Over-pruning basic block – miss solvable path
  - Grouping multiple executions
  - Context sensitivity – If block are in different branches

IMPLEMENTATION

Component	Lines of code
Concolic execution core	12,528 LoC of C++
Expression generation	1,913 LoC of C++
System call abstraction	1,577 LoC of C++
Hybrid fuzzing	565 LoC of Python

- Intel Pin used for emulation
  - API that allows context information such as register contents to be passed to the injected code as parameters

QSYM – REAL WORLD SCALABLE

- Apply QSYM to programs large in size and previously fuzzed
- 13 new unknown bugs found in software
- Google's OSS-Fuzz generated 10 trillion test inputs a day for a few months to fuzz these applications
  - QSYM ran them for three hours using a single workstation
- Driller – Hybrid Fuzzer (test cases)

COMPARISON

```
1  // @libavcodec/x86/mpegvideodsp.c:58 (ffmpeg 3.4)
2  if ( ((ox ^ (ox + dxw))
3        | (ox ^ (ox + dxh))
4        | (ox ^ (ox + dxw + dxh))
5        | (oy ^ (oy + dyw))
6        | (oy ^ (oy + dyh))
7        | (oy ^ (oy + dyw + dyh))) >> (16 + shift)
8       || (dxx | dxy | dyx | dyy) & 15
9       || (need_emu && (h > MAX_H || stride > MAX_STRIDE)))
10 { ... return; }
11 // the bug is here
```

- OSS – Fuzz (2 years)
- QSYM generates test case to reach this bug

LIMITATIONS

- Specialized to test on x86 architecture
- Other executors using IR can be ran on other architectures

CONCLUSION

- QSYM is a hybrid fuzzing model that is scalable to real world applications
- Outperforms current models for bug finding