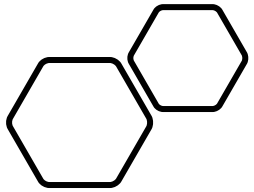
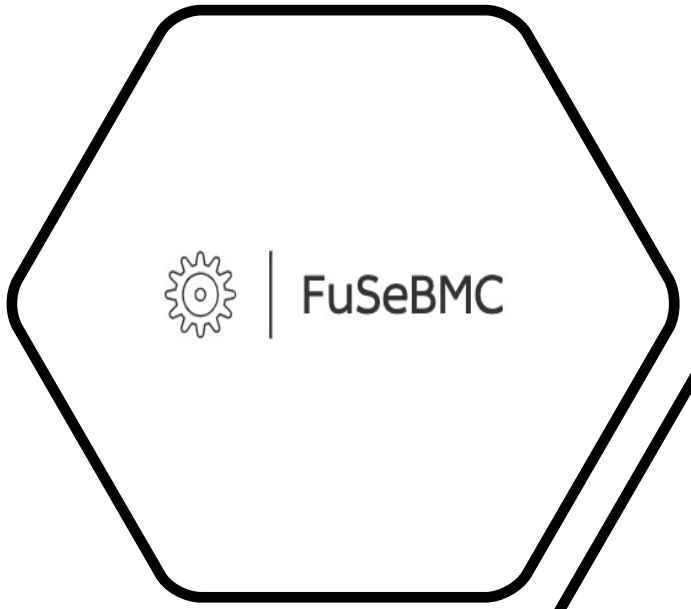
MANCHESTER 1824

The University of Manchester

FuSeBMC v.4: Smart Seed Generation for Hybrid Fuzzing

Author: Kaled Alshmrany
Kaled.alshmrany@manchester.ac.uk
Co-Authors: Mohannad Aldughaim, Ahmed Bhayat & Lucas C. Cordeiro





The Outline

- FuSeBMC v4 Team
- Motivation
- FuSeBMC v4 framework
- Seed generation
- Tracer
- Comparison results
- Software Project
- Awards

FuSeBMC Team

Mr. Kaled Alshmrany

FuSeBMC

Dr. Lucas Cordeiro

SCorCH

SCorCH team

ESBMC

ESBMC team

Institute of Public Administration

Motivation

Software testing is one of the most crucial phases in software development.
Tests often expose critical bugs in software applications.

- Exp:

Bounded Model Checking

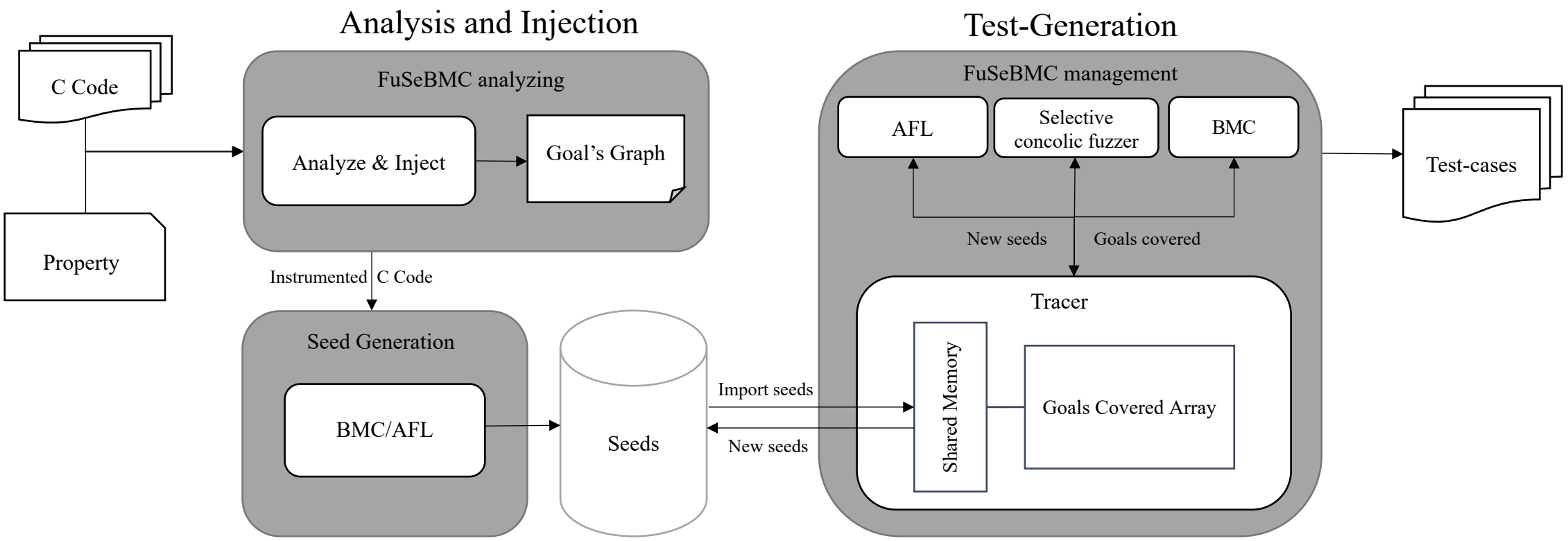
Fuzzing

- However, there is still a shortcoming in detecting these bugs due to the inability to cover large areas in the target code.
- Desire to develop our performance in Cover-Branches better than (FuSeBMC v3 achieved 4th place).

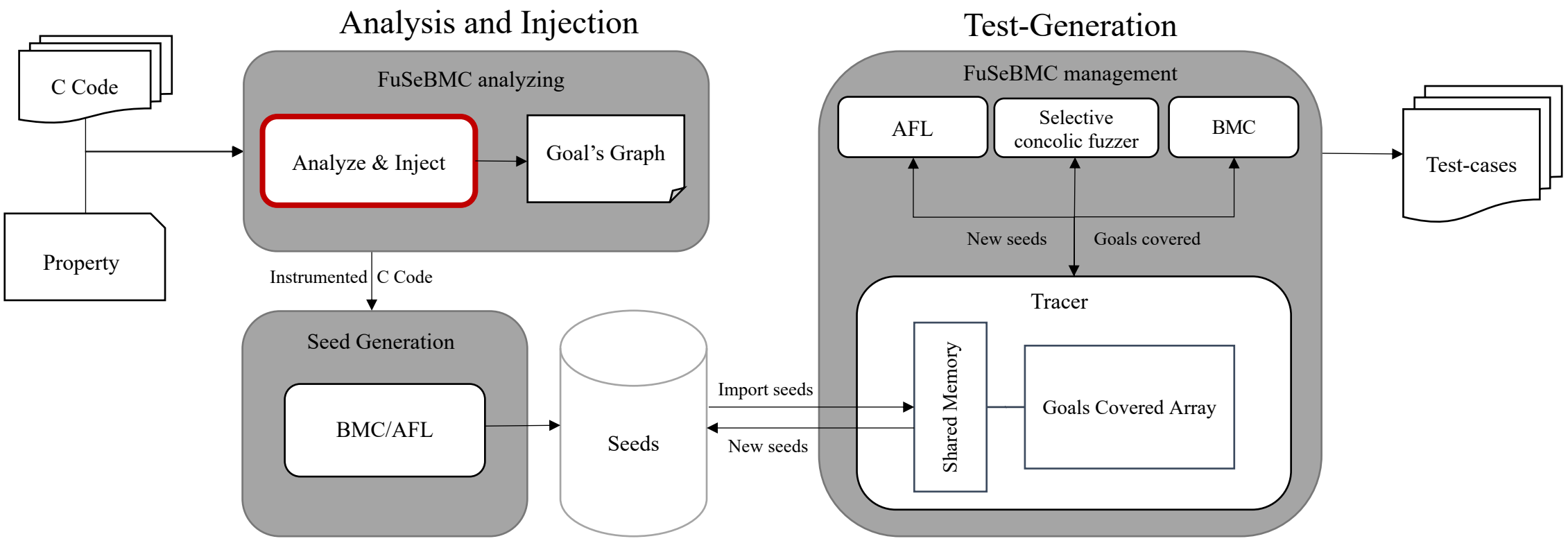
FuSeBMC

We propose FuSeBMC v4, a test generator that relies on smart seeds to improve the hybrid fuzzer to achieve high C programs coverage.

FuSeBMC Framework



FuSeBMC Framework



FuSeBMC Framework

```c
#include <assert.h>
void reach_error() { assert(0); }

bool check(int a, int b, int c, int x) {
  return (a*x*x + b*x + c == 0);
}

int main() {
  int a = __VERIFIER_nondet_int();
  int b = __VERIFIER_nondet_int();
  int c = __VERIFIER_nondet_int();
  if(b*b >= 4*a*c) {
    while(1) {
      int x = __VERIFIER_nondet_int();
      if(x <= 0 || x > 100)
        reach_error();
      if(check(a, b, c, x))
        return 0;
    }
  }
  else
    reach_error();

return 0;
}
```

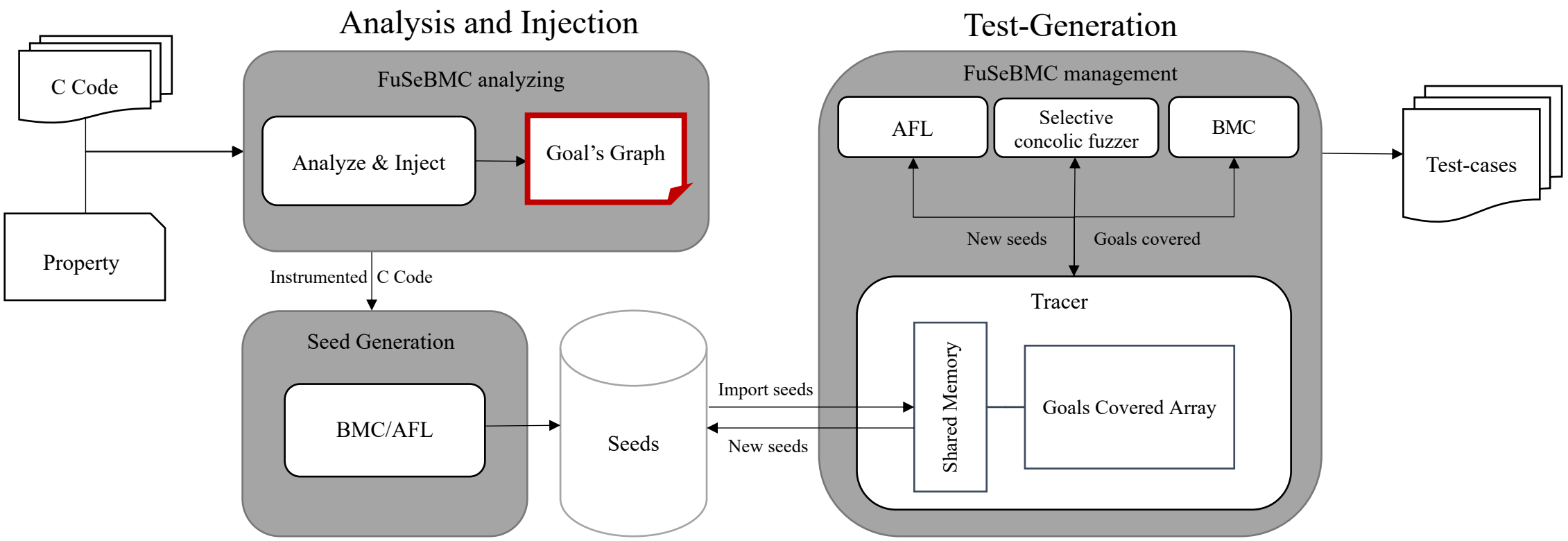
```c
#include <assert.h>
void reach_error() {
  GOAL_1:;
  assert(0);
}

bool check(int a, int b, int c, int x) {
  GOAL_2:;
  return (a*x*x + b*x + c == 0);
}

int main() {
  GOAL_0:;
  int a = __VERIFIER_nondet_int();
  int b = __VERIFIER_nondet_int();
  int c = __VERIFIER_nondet_int();
  if(b*b >= 4*a*c) {
    GOAL_4:;
    while(1) {
      GOAL_6:;
      int x = __VERIFIER_nondet_int();
      if(x <= 0 || x > 100) {
        GOAL_8:;
        reach_error();
      }
      if(check(a, b, c, x)) {
        GOAL_9:;
        return 0;
      }
    }
    GOAL_7:;
  }
  else {
    GOAL_5:;
    reach_error();
  }
  GOAL_3:;
  return 0;
}
```

FuSeBMC Framework

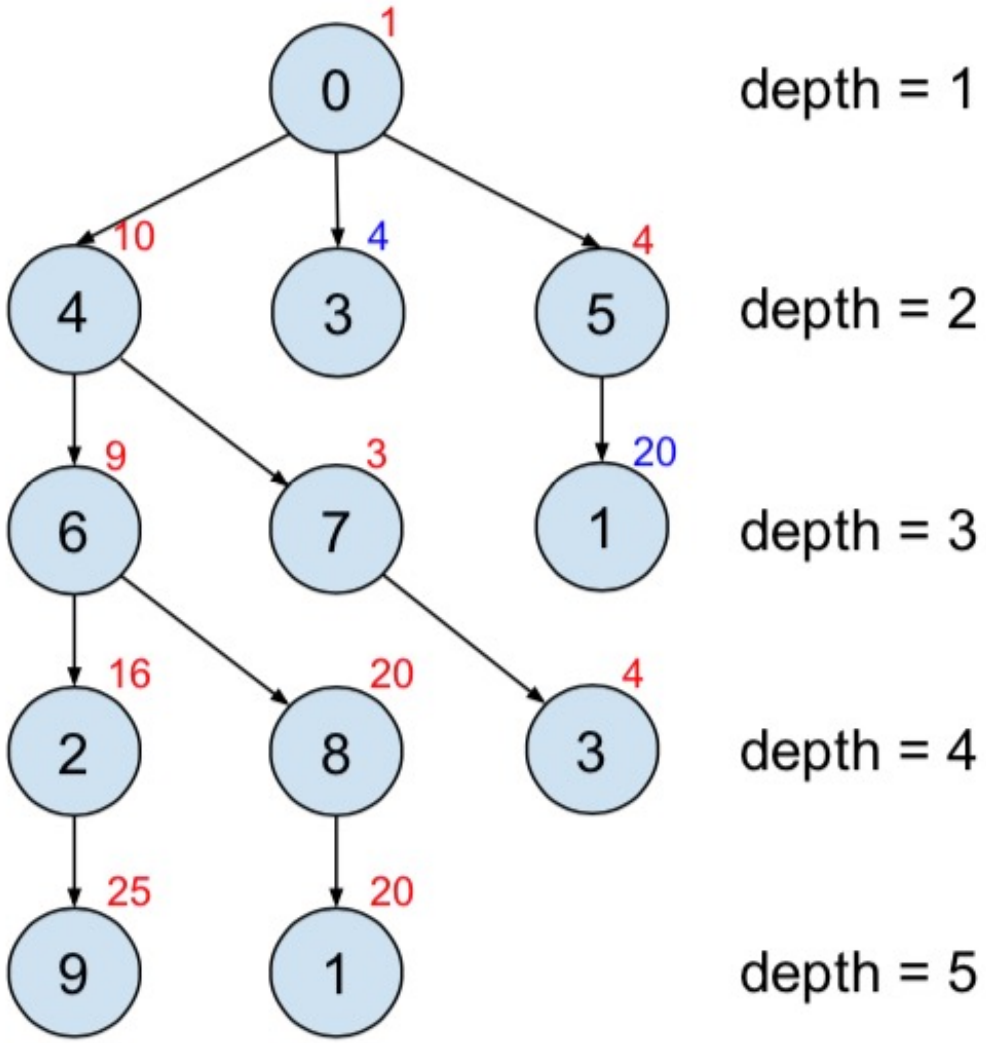
```
1  #include <assert.h>
2  void reach_error() { assert(0); }
3
4  bool check(int a, int b, int c, int x) {
5    return (a*x*x + b*x + c == 0);
6  }
7
8  int main() {
9    int a = __VERIFIER_nondet_int();
10   int b = __VERIFIER_nondet_int();
11   int c = __VERIFIER_nondet_int();
12   if(b*b >= 4*a*c) {
13     while(1) {
14       int x = __VERIFIER_nondet_int();
15       if(x <= 0 || x > 100)
16         reach_error();
17       if(check(a, b, c, x))
18         return 0;
19     }
20   }
21   else
22     reach_error();
23
24 return 0;
25 }
```

```
1  #include <assert.h>
2  void reach_error() {
3    GOAL_1:;
4    assert(0);
5  }
6
7  bool check(int a, int b, int c, int x) {
8    GOAL_2:;
9    return (a*x*x + b*x + c == 0);
10 }
11
12 int main() {
13   GOAL_0:;
14   int a = __VERIFIER_nondet_int();
15   int b = __VERIFIER_nondet_int();
16   int c = __VERIFIER_nondet_int();
17   if(b*b >= 4*a*c) {
18     GOAL_4:;
19     while(1) {
20       GOAL_6:;
21       int x = __VERIFIER_nondet_int();
22       if(x <= 0 || x > 100) {
23         GOAL_8:;
24         reach_error();
25       }
26       if(check(a, b, c, x)) {
27         GOAL_9:;
28         return 0;
29       }
30     }
31     GOAL_7:;
32   }
33   else {
34     GOAL_5:;
35     reach_error();
36   }
37   GOAL_3:;
38   return 0;
39 }
```

FuSeBMC Framework



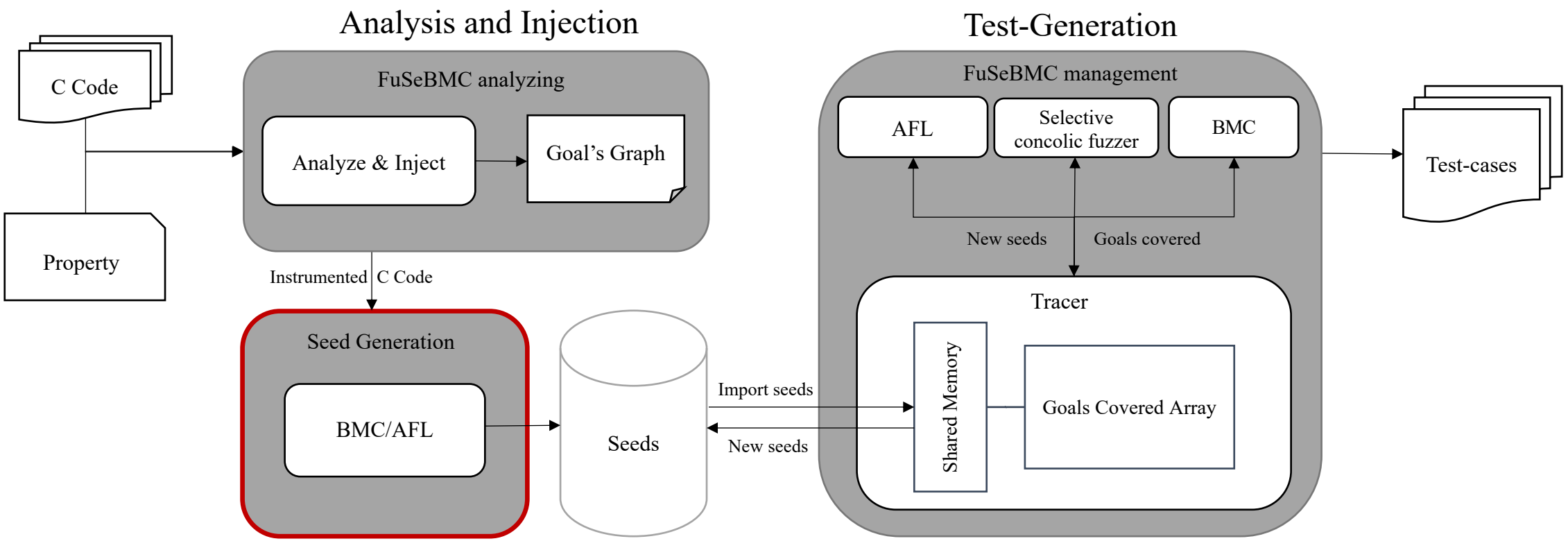
FuSeBMC Framework




```
 1 #include <assert.h>
 2 void reach_error() {
 3   GOAL_1:;
 4   assert(0);
 5 }
 6
 7 bool check(int a, int b, int c, int x) {
 8   GOAL_2:;
 9   return (a*x*x + b*x + c == 0);
10 }
11
12 int main() {
13   GOAL_0:;
14   int a = __VERIFIER_nondet_int();
15   int b = __VERIFIER_nondet_int();
16   int c = __VERIFIER_nondet_int();
17   if(b*b >= 4*a*c) {
18     GOAL_4:;
19     while(1) {
20       GOAL_6:;
21       int x = __VERIFIER_nondet_int();
22       if(x <= 0 || x > 100) {
23         GOAL_8:;
24         reach_error();
25       }
26       if(check(a, b, c, x)) {
27         GOAL_9:;
28         return 0;
29       }
30     }
31     GOAL_7:;
32   }
33   else {
34     GOAL_5:;
35     reach_error();
36   }
37   GOAL_3:;
38   return 0;
39 }
```

If = 5
Function = 4
Loops = 3
else = 2
After Loops = 1

FuSeBMC Framework



Seed Generation

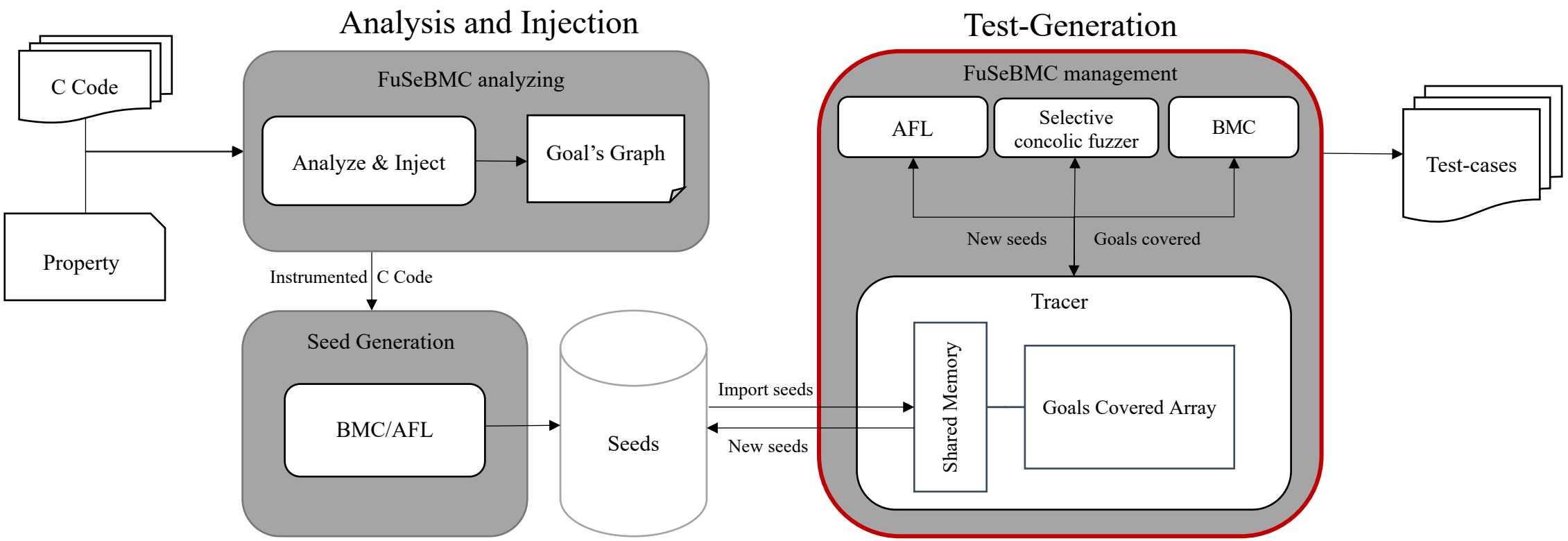
- Limiting loop bounds.
- Assuming a narrow range of values for input variables.
- Ranking the the test cases generated by these engines.
- Adding the selected seeds to the seed store.

- The impact of a test case is measured using two metrics:
  - The number of labels covered uniquely by that test case.
  - The maximum program depth achieved by the test case.

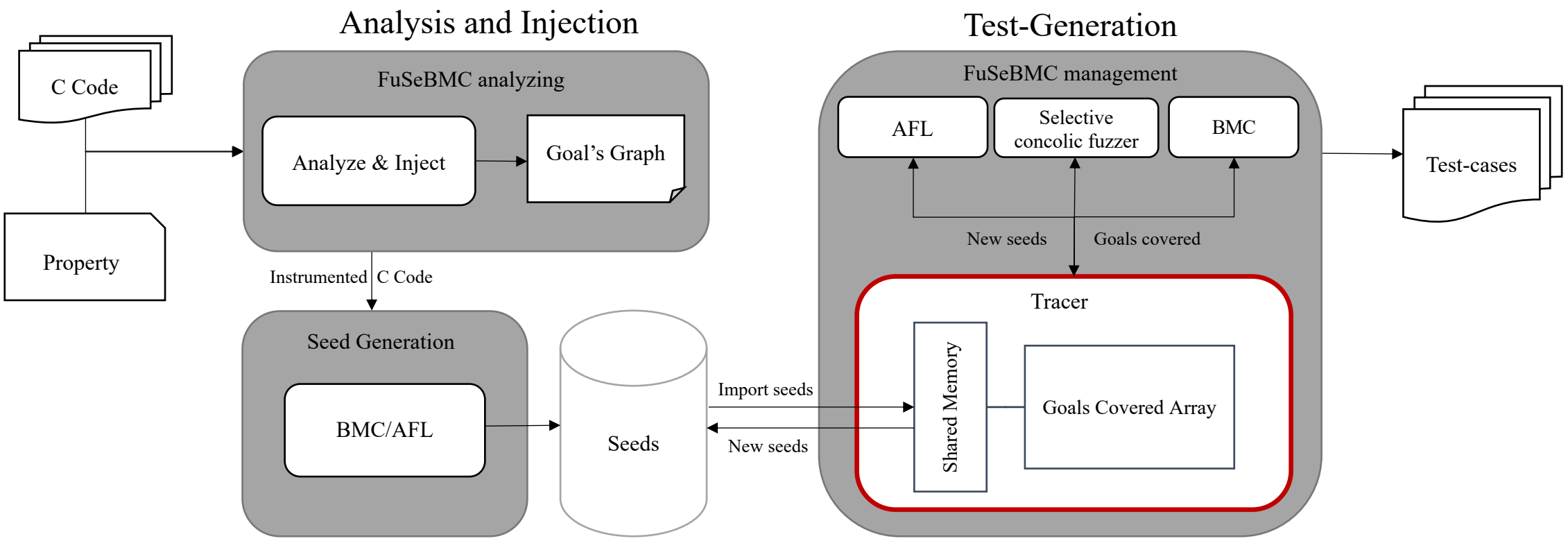
Seed Generation

- BMC is particularly effective at seed generation to discover test cases that circumvent complex mathematical guards.
- A fuzzer on its own, randomly mutating a seed, struggles to explore program sections occurring behind complex guards.

FuSeBMC Framework



FuSeBMC Framework

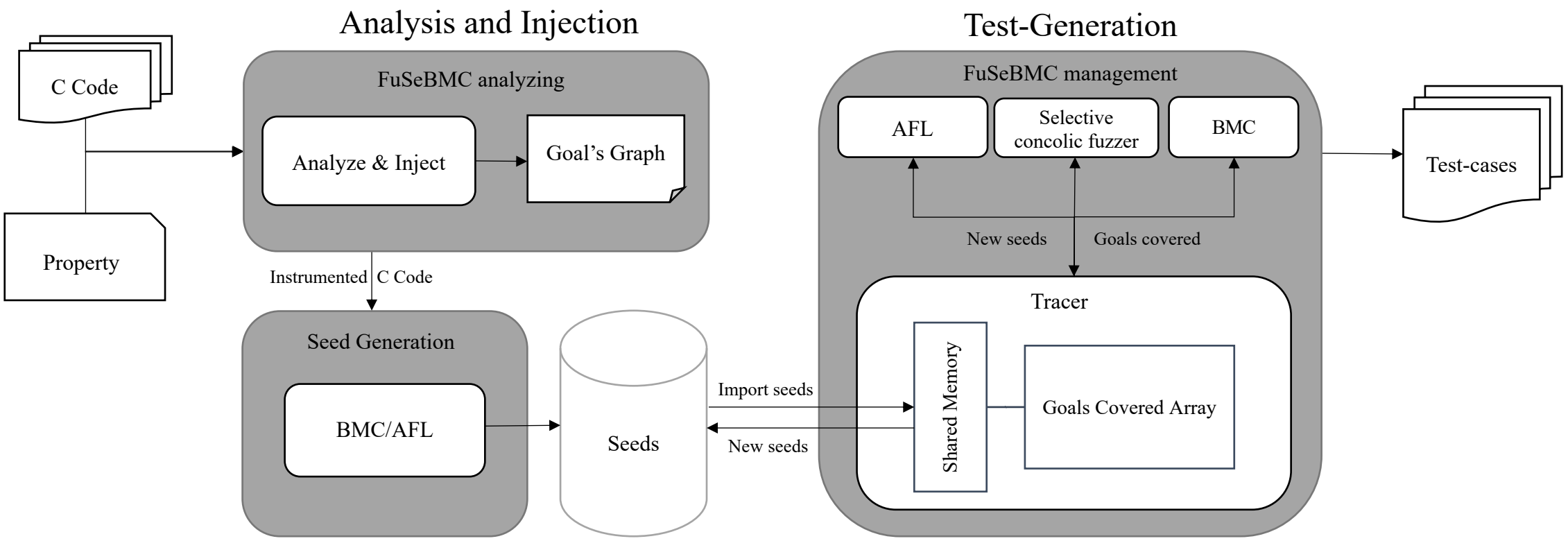


Tracer

- Coordinates the various engines through the use of shared memory.
- Has 2 component:
  - Goals covered array
  - Maintains effective seeds

- Why Tracer enhance the performance?
  - Reduces the number of BMC executions - because BMC can be slow and resource-intensive.
  - Monitors and evaluates the engines, adding those with the highest impact to the seed store.
  - The seed store is dynamically updated as the analysis progresses.

FuSeBMC Framework



The improvement achieved by FuSeBMC v4 in comparison to FuSeBMC v3 for the Cover-Error category

Subcategory	% success		Improvement Δ%
	FuSeBMC v4	*FuSeBMC* v3	
Arrays	99%	93%	6%
BitVectors	100%	100%	0%
ControlFlow	100%	25%	75%
ECA	72%	44%	28%
Floats	100%	97%	3%
Heap	95%	80%	14%
Loops	93%	83%	10%
ProductLines	100%	0%	100%
Recursive	95%	95%	0%
Sequentialized	95%	94%	1%
XCSP	88%	90%	-2%
BusyBox	15%	0%	15%
DeviceDrivers	0%	0%	0%
Average value	81%	67%	14%

The improvement achieved by FuSeBMC v4 in comparison to FuSeBMC v3 for the Cover-Branches category

Subcategory	% coverage		Improvement Δ%
	FuSeBMC v4	*FuSeBMC v3*	
Arrays	82%	71%	11%
BitVectors	80%	60%	20%
ControlFlow	64%	22%	42%
ECA	37%	17%	20%
Floats	54%	46%	8%
Heap	73%	62%	11%
Loops	81%	71%	10%
ProductLines	29%	0%	0%
Recursive	85%	68%	18%
Sequentialized	87%	76%	11%
XCSP	90%	82%	8%
Combinations	61%	7%	53%
BusyBox	34%	1%	32%
DeviceDrivers	20%	12%	8%
SQLite-MemSafety	4%	0%	4%
Termination	92%	87%	5%
Average value	61%	45%	16%

Software Project

FuSeBMC source code is written in C++ and Python; it is available for download on GitHub. Also, the instructions for using the tool FuSeBMC are given in the file README.

```
kaled@kaled-VirtualBox:~/Desktop/FuSeBMC_v3.6.6$ ./fusebmc.py -s incr -p properties
/coverage-branches.prp sv-benchmarks/c/array-tiling/skippedu.c
```

Awards

FuSeBMC received three significant awards from the 4th International Competition on Software Testing (Test-Comp 2022) organised by the European Joint Conferences on Theory and Practice of Software (ETAPS).

- FuSeBMC got first place in the most critical category of Test-Comp: **Cover-Error** (find a test that covers a bug).

- FuSeBMC got first place in the category of Test-Comp: **Cover-Branches** (find a test that covers a branch).

- FuSeBMC earned first place in Test-Comp's overall category.

Want to Try it?

Find out more about FuSeBMC at :
https://github.com/kaled-alshmrany/FuSeBMC

Test-Comp'22 paper: "FuSeBMC v4: Smart Seed Generation for Hybrid Fuzzing (Competition Contribution)"

kaled.alshmrany@postgrad.manchester.ac.uk

Thank you…