TestCov

Test Execution and Coverage Measurement in Test-Comp

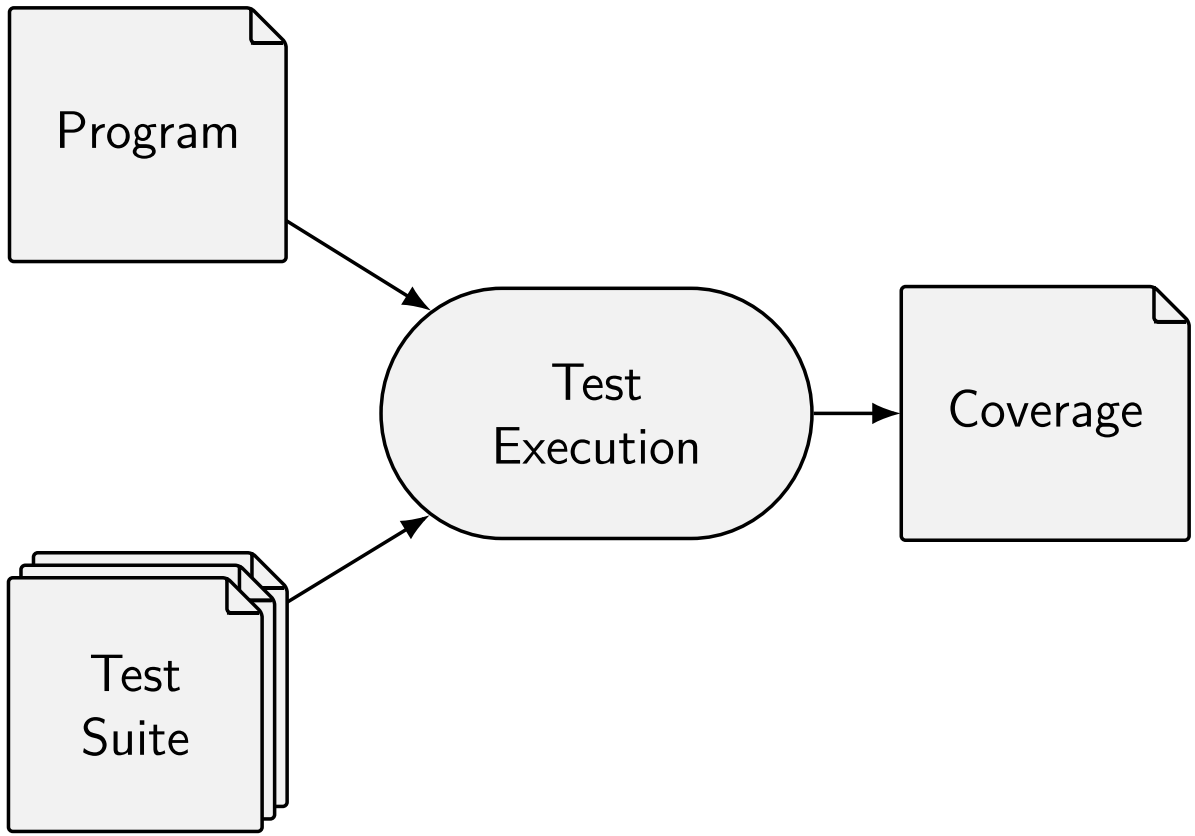
Thomas Lemberger

LMU Munich, Germany

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

Our Starting Point



The Issue

```
1  #include <stdio.h>
2  #include <unistd.h>
3  extern char input();
4
5  int main() {
6    char x = input();
7    if (x == 'a') {
8      while (1) {
9          fork();
10     }
11   } else {
12     remove("important.txt");
13     if (access("important.txt", F_OK) != -1) {
14       return 1;
15     }
16   }
17 }
```

The Issue

```
1  #include <stdio.h>
2  #include <unistd.h>
3  extern char input();
4
5  int main() {
6    char x = input();
7    if (x == 'a') {
8      while (1) {
9        fork();
10     }
11   } else {
12     remove("important.txt");
13     if (access("important.txt", F_OK) != -1) {
14       return 1;
15     }
16   }
17 }
```

The Issue

```
1  #include <stdio.h>
2  #include <unistd.h>
3  extern char input();
4
5  int main() {
6    char x = input();
7    if (x == 'a') {
8      while (1) {
9        fork();
10     }
11   } else {
12     remove("important.txt");
13     if (access("important.txt", F_OK) != -1) {
14       return 1;
15     }
16   }
17 }
```

The Issue

```
1  #include <stdio.h>
2  #include <unistd.h>
3  extern char input();
4
5  int main() {
6    char x = input();
7    if (x == 'a') {
8      while (1) {
9        fork();
10     }
11   } else {
12     remove("important.txt");
13     if (access("important.txt", F_OK) != -1) {
14       return 1;
15     }
16   }
17 }
```

- Goal: Achieve 100 % branch coverage
- But: We don't want to use our system to execute the test suite that achieves that.

Existing Solutions to Robust Execution

- Virtual Machines
- Containerization (Docker etc.)

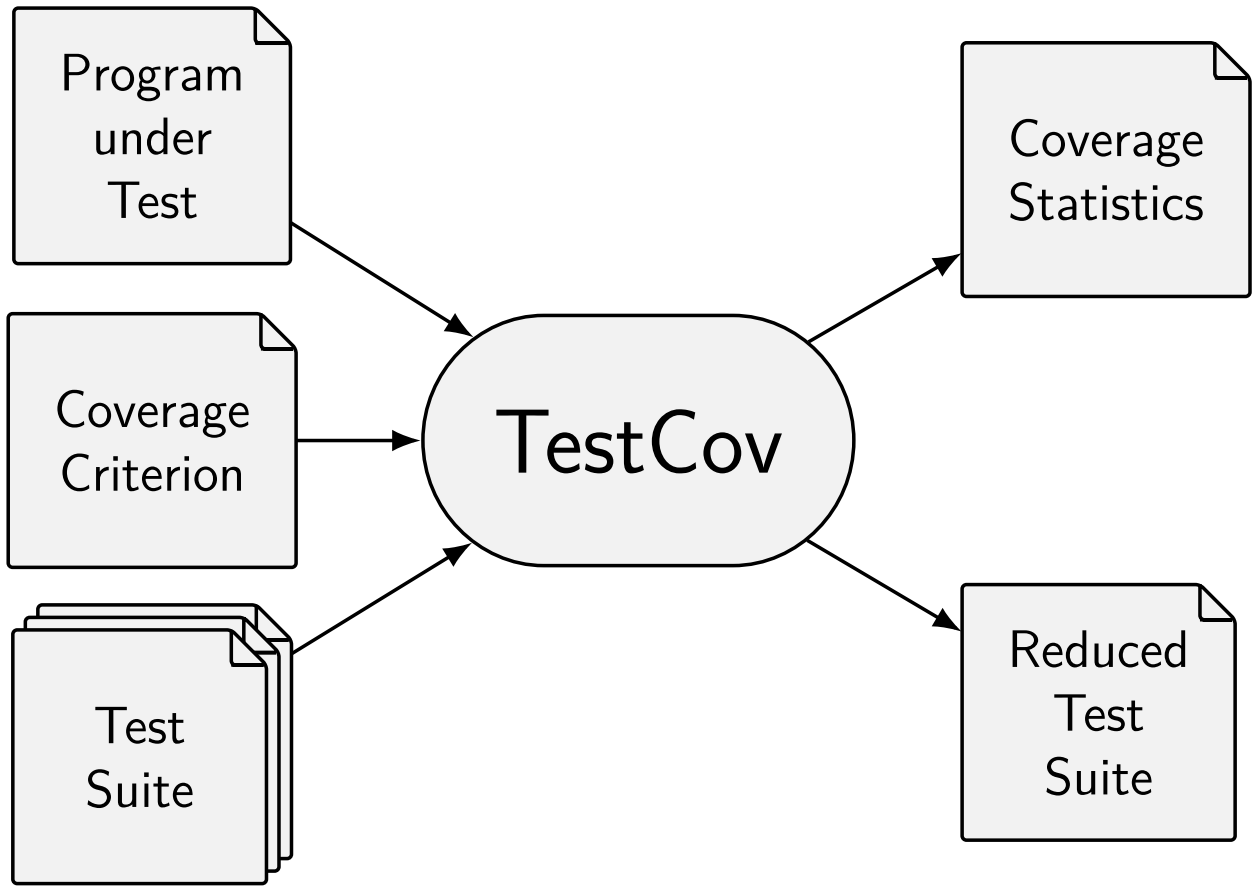
$\Rightarrow$ Potentially large overhead

$\Rightarrow$ Manual setup

$\Rightarrow$ Setups consist of multiple tools

$\Rightarrow$ Require superuser privileges

Our Solution



Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers
- Each individual test execution isolated

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers
- Each individual test execution isolated
- Protection against:

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers
- Each individual test execution isolated
- Protection against:
  - Resource exhaustion

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers
- Each individual test execution isolated
- Protection against:
  - Resource exhaustion
  - File system modifications

Robust Test Execution

- Test isolation through Linux kernel features (BenchExec)
  https://github.com/sosy-lab/benchexec/
  - Control Groups (CGroups)
  - Containers
- Each individual test execution isolated
- Protection against:
  - Resource exhaustion
  - File system modifications
  - Dependencies between tests

Coverage Measurements

- `lcov` + `gcov` for line coverage

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:
  - Test success

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:
  - Test success
  - Individual test coverage

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:
  - Test success
  - Individual test coverage
  - Accumulated test coverage (after each execution)

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`

- Produced data:
  - Test success
  - Individual test coverage
  - Accumulated test coverage (after each execution)
  - Resource consumption per test execution

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:
  - Test success
  - Individual test coverage
  - Accumulated test coverage (after each execution)
  - Resource consumption per test execution
  - `.json` data + `.svg` plot

Coverage Measurements

- `lcov` + `gcov` for line coverage
- Test-Comp coverage computed through program instrumentation `GOAL_n:;`
- Produced data:
  - Test success
  - Individual test coverage
  - Accumulated test coverage (after each execution)
  - Resource consumption per test execution
  - `.json` data + `.svg` plot
  - Reduced test suite

TestCov available open source (Apache 2.0):

https://gitlab.com/sosy-lab/software/test-suite-validator/

Appendix

References

[1] D. Beyer, S. Löwe, and P. Wendler.
Reliable benchmarking: Requirements and solutions.
Int. J. Softw. Tools Technol. Transfer, 21(1):1–29, 2019.

Test-Suite Format

- XML-based
- Two components:
  1. `metadata.xml`
  2. one XML-file per test case
     - Sequence of test inputs
- Handled as zip archive

Metadata

```
<?xml version="1.0"?>
<!DOCTYPE test-metadata PUBLIC "+//IDN sosy-lab.org//DTD test-format te
<test-metadata>
  <sourcecodelang>C</sourcecodelang>
  <producer>Testsuite Validator v2.0</producer>
  <specification>CHECK(FQL(cover EDGES(@CONDITIONEDGE)))</specificatio
  <programfile>example.c</programfile>
  <programhash>eeecda9cbf27c43c9017fa00dd900c19a5ec18d46303f59a6e0357db78
  <entryfunction>main</entryfunction>
  <architecture>32bit</architecture>
  <inputtestsuitefile>original-suite.zip</inputtestsuitefile>
  <inputtestsuitehash>11911d658dcfbf8501390bf0faa96eb193b11bb1</inputtestsui
  <creationtime>2019-06-19T14:17:34Z</creationtime>
</test-metadata>
```

Test Case

```
<?xml version="1.0"?>
<!DOCTYPE testcase PUBLIC "+//IDN sosy−lab.org//DTD test−format testcase
<testcase>
  <input>'b'</input>
  <input>10</input>
  <input>0x0f</input>
</testcase>
```