Representation and Analysis of Code Coverage Reports from CPAchecker

Vladimir Gratinsky, Evgeny Novikov, Ilja Zakharov
ISP RAS

CPA'21, Online, September 30, 2021

Challenges at Verification of Particular Programs

- Unclear program environments and requirements should be formalized
- Verification should not fail and run out of time/memory
- There may be no false alarms due to any reason
- There may be no missed bugs due to verification tools
- Code coverage should be 100%+

```
int array[10];
int func(int index) {
    return array[index];
}
```

- Verification results should be represented in a user-friendly way

Contribution of Code Coverage to Formalization of Unclear Program Environments and Requirements

Original program source file

```
int func(const char *string) {
    ...
    if (strlen(string) > 7)
        return -1;
    ...
}
```

Environment model 1

```
void entry_point(void) {
    func(“Hello”);
}
```

Environment model 2

```
int entry_point(void) {
    func(“World”);
}
```

Contribution of Code Coverage to Formalization of Unclear Program Environments and Requirements

Original program source file

```
int func(const char *string) {
    ...
    if (strlen(string) > 7)
        return -1;
    ...
}
```

Environment model 1

```
void entry_point(void) {
    func(“Hello”);
}
```

Environment model 2 (+requirement specification)

```
int entry_point(void) {
    if (func(“Hello, World”) != -1)
        ldv_assert();
}
```

Contribution of Code Coverage to Detection of Most Complicated Parts of Programs

Original program source file

```
void func(unsigned int size) {
    int i;
    ...
    682 stops for total time 20 s
    for (i = 0; i < size; i++)
        ...
    ...
}
```

Code coverage legend

4000	2000	1	0

Environment model

```
int entry_point(void) {
    func(1000);
}
```

Contribution of Code Coverage to Detection of Most Complicated Parts of Programs

Original program source file

```
void func(unsigned int size) {
    int i;
    ...
    100 stops for total time 2 s
    for (i = 0; i < size; i++)
        ...
    ...
}
```

Code coverage legend

Environment model

```
int entry_point(void) {
    func(100);
}
```

Contribution of Code Coverage to Reasoning about False Alarms, Missed Bugs and 100%+

Original program source file

```
int array[10];
int func(int index) {
    return array[index];
}
```

Requirement specification

```
int entry_point(void) {
    int index = ldv_random_int(0, 9);
    int value = ldv_undef_int();

    array[index] = value;
    if (func(index) != value)
        /* ASSERT ... */
        ldv_assert();
}
```

Contribution of Code Coverage to Reasoning about False Alarms, Missed Bugs and 100%+

Original program source file

```
int array[10];
int func(int index) {
    index = {0..9}
    return array[index];
}
```

Requirement specification

```
int entry_point(void) {
    int index = ldv_random_int(0, 9);
    int value = ldv_undef_int();

    array[index] = value;
    if (func(index) != value)
        /* ASSERT ... */
        ldv_assert();
}
```

Processing of CPAchecker's Code Coverage Reports in Klever

- Relating code coverage with corresponding original source files and models rather than with CIL files
- Calculating statistics for each individual source file as well as for each subdirectory of the program source tree
- Merging code coverage reports issued for individual verification tasks to get a single code coverage report for the whole program
- Converting code coverage reports to the more appropriate form for their visualization*

* *https://klever.readthedocs.io/en/latest/dev.html#code-coverage-format*

Demo of Real Code Coverage Examples*

- Example 1 – incomplete environment model
- Example 2 – overcomplicated environment model
- Example 3 – code coverage for unsafe
- Example 4 – total code coverage and statistics

* Examples are not publicly available!

What’s Next?

- Get more useful data from different CPAchecker’s analysis
- Fix relation of provided data with sources
- Standardize code coverage reports for automatic software verification tools