Karlsruher Institut für Technologie

Institute of Information Security and Dependability (KASTEL)
Application-oriented Formal Verification
Prof. Dr. Bernhard Beckert
https://formal.kastel.kit.edu

Bachelor Thesis, Master Thesis, Praxis der Forschung
Change-Aware Test Impact Analysis and Repair

Motivation

Maintaining a large number of unit test cases in industrial software products is costly due to continuous software evolution. When the source code changes through additions, removals, or modifications, it often becomes unclear which test cases need adaptation and to what extent. Moreover, unit tests observe only isolated input–output pairs and therefore provide weak guarantees of functional correctness and behavioural invariants.

This work targets change-aware test impact analysis and specification extraction from existing tests to (i) identify exactly which tests are affected by a change, (ii) automatically propose repairs or evolutions of tests, and (iii) derive candidate pre/postconditions that can be checked beyond the enumerated test inputs.

Illustrative Example: Code Evolution vs. Test Adequacy

(1) Original Version

Code:

```
int avg(int a,int b){
  return (a+b)/2;
}
```

Test:

```
assertEquals(3, avg(3,4));
```

(2) Updated Version

Code:

```
int avg(int a,int b){
  return (a+b)>>1;
}
```

Observation:

- All existing tests still pass
- Silent semantic drift not detected

Example:
Old: avg(-3,-4) = -3
New: avg(-3,-4) = -4 (different)

(3) Change-Aware Approach

Inferred Postcondition:

$$2r \leq a + b < 2r + 2$$

Hidden behaviour change becomes visible
Test Evolution and Repairs:

- Add missing tests
- Generalise tests
- Suggest code refinement

Outcome:
Tests become a **specification-aware asset** that survives evolution.

Benefits:

- Reduce maintenance costs by automatically evolving tests and deriving specifications.
- Enable formal verification tools to prove properties beyond test coverage.
- Improve documentation and preservation of intended program behaviour over time.
- Bring formal methods closer to real-world development by integrating with continuous testing.

Prerequisites

A background in software engineering, testing, or formal methods is recommended. Knowledge of a programming language and its testing frameworks is required. Familiarity with specification languages is advantageous but not mandatory.

Contact
Dr. Tianhai Liu
tianhai.liu@kit.edu | Office: 50.34 R227