March 2023

Automated Testing of Software with Adequate Coverage of Input Space

Matt Kenison

Justin Bagwell

Mike Meade

Automated Testing of Software with Adequate Coverage of Input Space

ABSTRACT

Some software products have large input spaces, with potentially an exponentially large number of parameter combinations. It is infeasible to exhaustively test such software products. This disclosure describes techniques to automatically generate input space coverage for a software product by generating test cases from a rules grammar while maintaining coverage over the lifetime of the product. The various components include a grammar for describing the input space; a test engine to accept workloads and input parameters and to generate test cases that run workloads against input parameters; a software module that computes the number of interacting parameters to test and efficient combinations thereof to make a tractable number of test inputs; etc.

KEYWORDS

- Software testing
- *t*-way testing
- Combinatorial testing
- Test coverage
- Covering array
- Test case generation

BACKGROUND

Some software products have large input spaces, with potentially an exponentially large number of parameter combinations. It is infeasible to exhaustively test such software products. A fixed subset of parameters is insufficient for comprehensive coverage of a software product, especially one under active development, since the input space changes as the product features change. Randomly parameterizing tests can enable a higher degree of combinatorial coverage but does not do so deterministically. Also, randomly parameterized tests do not guarantee comprehensive coverage.

Existing techniques, e.g., *t*-way parametrization, iterative sieves, etc., largely focus on algorithmic parameterization, optimizing permutations, and reducing the number of test cases needed to cover an input space. There is no comprehensive solution for automatically generating input space coverage or maintaining it over the lifetime of a software product.

DESCRIPTION

This disclosure describes techniques to automatically generate input space coverage for a software product by generating test cases from a rules grammar while maintaining coverage over the lifetime of the product.

The various components of the architecture are described below.

- A grammar for describing the input space of a software product or module. The terminal symbols of the grammar are dimension values. The non-terminal symbols are dimension subsets. Subsets can be statically defined, generated from data, or computed from lambda functions.
- A test engine, which
  - accepts test workloads for one or more dimensions;
  - accepts a set of dimensional values as input parameters;
  - defines a test oracle for each workload; and
  - generates a test case that runs each workload against input parameters and validates rules using the test oracle.
- A software module, which:

- specifies the number *t* of interacting parameters to test and a desired confidence level of the results;
- defines a language using the grammar to derive the possible dimensional values for each parameter;
- computes efficient combinations of the *t* dimensional values into a tractable number of test inputs;
- runs, for each combination of dimensional values, each test case, passing the combination of values as input to the test engine;
- statistically processes failing tests to produce new combinations of the input parameters and repeats tests until the desired confidence level is achieved; and
- reports test results that correlate test failures to particular combinations of dimensional values with which those tests were run.

Fig. 1 illustrates an architecture for automated testing of software.

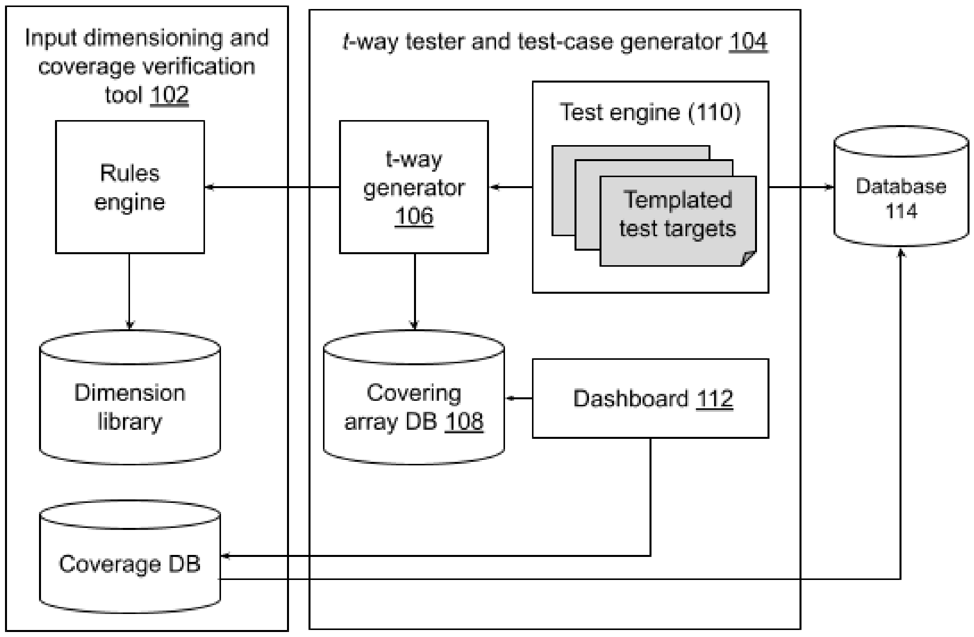

Fig. 1: Architecture for automated testing of software

Fig. 1 illustrates an example architecture for automated testing of software input. A software tool (102) (e.g., as described in greater detail in [1] or similar), defines input rules or dimensions, and gathers and verifies test coverage. A *t*-way tester and test-case generator (104) can generate test cases from a rules grammar. It includes:

- ***t*-way generator (106):** A module that generates a covering array of all valid *t*-way combinations of parameters defined in the configuration of the software tool (102). The *t*-way generator can be based on, for example, open source software (e.g., as described in [6]) proven in life-critical applications such as medical safety, aviation, etc. Alternatively, the *t*-way generator can be written from custom software based on covering-array generators (e.g., as described in [7]). The *t*-way generator writes the generated combinations to a covering array database (108) that can be queried by a separate test runner.
- **Covering-array database (108):** A database that stores individual *t*-way combinations. It can store more combinations than are used in testing without side effects, and can generate combinations for varying levels of *t*. The expected coverage can be directly calculated from this data without running tests, so if 100% coverage is not required, an informed risk tradeoff can be made.
- **Test engine (110):** A module that generates and runs tests to provide test coverage. For each combination of parameters, the test engine creates a workflow by selecting applicable test targets, applies the specified combinatorial values, and schedules the test workflow. Tests can take advantage of generic workflow scheduling and throttling to ensure reliability. Test targets can vary based on selected parameters.

- **Visualization:** A dashboard (112) that graphically displays coverage results and enables the user to deep-dive into failures. The dashboard can show the relative coverage of all *N*-way combinations (including the case $N > t$, which enables the making of an informed decision about the appropriate value for *t*).

The *t*-way tester and test-case generator can interact with an external database (114) that enables users to query aspects of testing and coverage. The dashboard and the other components (*t*-way generator, covering-array database, test engine) are minimally coupled to each other and have no external dependencies. The techniques generally apply to the testing of software products with a combinatorial number of input possibilities, e.g., virtual machine configurations for cloud computing, etc.

CONCLUSION

This disclosure describes techniques to automatically generate input space coverage for a software product by generating test cases from a rules grammar while maintaining coverage over the lifetime of the product. The various components include a grammar for describing the input space; a test engine to accept workloads and input parameters and to generate test cases that run workloads against input parameters; a software module that computes the number of interacting parameters to test and efficient combinations thereof to make a tractable number of test inputs; etc.

REFERENCES

[1] Kenison, Matt; Bagwell, Justin; Sampson, Tylor; and Meade, Mike. "Validating software functionality across combinations of runtime configurations," Technical Disclosure Commons, (December 12, 2022) available online at https://www.tdcommons.org/dpubs_series/5568

[2] D.R. Kuhn, I. Dominguez, R.N. Kacker and Y. Lei. "Measuring test quality with combinatorial coverage," available online at https://csrc.nist.gov/Projects/automated-combinatorial-testing-for-software/combinatorial-coverage-measurement/coverage-measurement accessed Mar. 12, 2023.

[3] Bonn, Joshua, Konrad Fögen, and Horst Lichter. "A framework for automated combinatorial test generation, execution, and fault characterization." In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 224-233. IEEE, 2019.

[4] Grindal, Mats, and Jeff Offcutt. "Input parameter modeling for combination strategies," available online at https://cs.gmu.edu/~offutt/rsrch/papers/ipmmodel.pdf accessed Mar. 12, 2023.

[5] D.R. Kuhn, R.N. Kacker, and Y. Lei. "Practical combinatorial testing," available online at https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf accessed Mar. 12, 2023.

[6] "USNISTGOV/combinatorial-testing-tools," available online at https://github.com/usnistgov/combinatorial-testing-tools accessed Mar. 12, 2023.

[7] Wagner, Michael, Kristoffer Kleine, Dimitris E. Simos, Rick Kuhn, and Raghu Kacker. "CAGEN: A fast combinatorial test generation tool with support for constraints and higher-index arrays." In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 191-200. IEEE, 2020.