Test Generation and Mutation Analysis of Energy Consumption using UPPAAL SMC and MATS

Jonatan Larsson *, Eduard Paul Enoiu*
*Mälardalen University, Sweden
jonatanlarsson8@hotmail.se, eduard.paul.enoiu@mdu.se

***Abstract*— Testing is an essential process for ensuring the quality of the software. Designing software with as few errors as possible in most embedded systems is often critical. Resource usage is a significant concern for proper behaviour because of the very nature of embedded systems. To design energy-efficient systems, approaches are needed to catch desirable consumption points and correct them before deployment. Model-based testing can reduce testing effort, one testing method that allows for automatic test generation. However, this technique has yet to be studied extensively for revealing resource usage anomalies in embedded systems development. UPPAAL SMC is a statistical model-checking tool that can model the system's resource usage. This paper shows the tooling needed to achieve this and experimental results on automated test generation and selection using mutation analysis in UPPAAL SMC and how this is applied to a Brake by Wire industrial system. The evaluation shows that this approach, which we call MATS, is applicable and efficient for energy-based test generation.**

***Index Terms*—energy consumption, mutation analysis, test generation, UPPAAL SMC**

I. Introduction

To ensure the development of reliable systems with fewer errors, the creation of effective extra-functional testing methods and tools is required. With manual testing being labour-intensive and error-prone, there is a growing need for efficiently created tests as the complexity and resource usage is increasing dramatically.

Model checkers have been used to automate the creation of tests. State-of-the-art model-checkers (i.e., UPPAAL SMC [1]) can be used to model the system's resource usage, thus enabling the generation of test cases that target such extra-functional requirements. This paper shows some experiments using UPPAAL SMC and how to use it to perform automated test generation for energy consumption.

Several recent studies [2], [3] have proposed methods for automatic test generation and a framework to perform fault detection analysis. Others have focused on using mutation testing for timed automata [4], [5]. In comparison with this line of research, our work is not aiming to describe how faults are deliberately injected into the specification model, but we present the necessary tooling to integrate mutation analysis and test generation with specific energy-aware models created in UPPAAL SMC.

This paper shows the practical tooling (i.e., the MATS tool[1]) needed to achieve test generation and mutation analysis of energy consumption and several experiments showing its usage. We show how to generate test cases in UPPAAL SMC by using its ability to simulate and evaluate the generated tests by measuring each test suite's ability to detect injected faults. Based on this method, one can reduce the test suite by selecting only the tests that maximize mutation coverage (i.e., the ratio of mutants killed to the total number of mutants). To evaluate this solution, we apply it to an industrial system: a Brake By Wire industrial prototype system. The results from the evaluation show that tests can be efficiently generated to achieve full mutation coverage on the system (around 277 seconds for the Brake By Wire models).

[1]The MATS tool is available as an open source solution at https://github.com/JLN93/MATS-Tool

II. UPPAAL SMC

UPPAAL [6] is a toolbox used to model and analyze real-time systems modelled as timed automata (TA), developed jointly by Uppsala and Aalborg University. UPPAAL SMC (Statistical Model Checker) [1] is an extension of UPPAAL that allows a user to simulate the system over many runs and has the ability to give approximate responses using statistical analysis and an estimate of the correctness of the property to be checked with a given confidence level. By employing such techniques, UPPAAL SMC avoids the exhaustive state-space search of usual symbolic model checking, hence providing benefits in terms of improved scalability. The SMC extension also supports priced timed automata models which improve the ability to model continuous resources such as energy. UPPAAL can compose Timed automata in parallel to form a network of TA to model complex systems.

In UPPAAL SMC queries are used to control the evaluation process and the query language UPPAAL SMC uses is Weighted Metric Temporal Logic (WMTL). The following properties can be verified with UPPAAL SMC:

- Probability Estimation: What is the probability of an event (i.e., a conjunction of predicates over the states of the TA network) to occur?
- Hypothesis Testing: Is the probability of an event greater or equal to a certain threshold?
- Probability Comparison: Is the probability of one event greater than the probability of another event?

  UPPAAL SMC also supports the ability to visualize a simulation of a system (or a set of overlapping simulations).

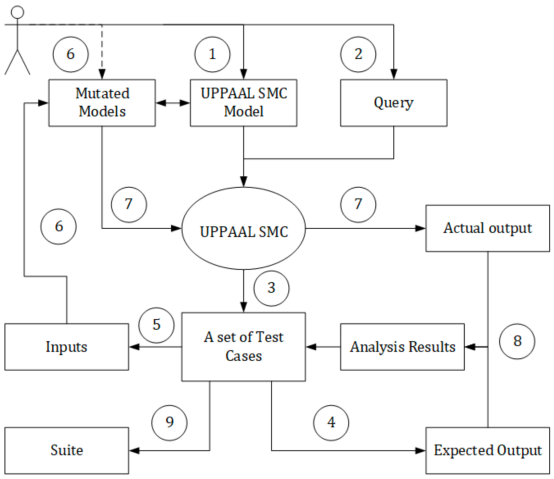


Fig. 1. An overview of the MATS approach for test generation and mutation analysis of energy consumption using UPPAAL SMC.

III. MATS Approach

In this section, we describe the test generation and mutation analysis approach that uses mutation analysis to automatically select test suites based on random system simulations. An overview of the approach is presented in Figure 1, where the numbered steps are the following:

- Step 1: The engineer creates a model, an abstraction of the SUT that is evaluated.
- Step 2: The engineer also formulates the query to guide the test case generation.
- Step 3: With the provided model and query, UPPAAL SMC randomly generates simulation traces, which represent our test cases and contain the set of inputs and the expected output.
- Step 4: From this, the test cases (i.e., inputs and output values) are extracted and used later in Step 8.
- Step 5: The generated input values are extracted and used for mutation analysis.
- Step 6: The sequence of inputs in each test case is automatically inserted in a set of mutated models that the engineer has created or generated.
- Step 7: The mutated models are simulated with the extracted inputs using UPPAAL SMC to generate new sets of actual energy outputs from the mutated models in order to measure the difference between the original and the mutated model.
- Step 8: The actual energy outputs extracted from the mutated models for each test case will be compared to the expected energy outputs in order to determine the test case's ability to kill (detect) any difference between the mutated models and the original (assumed correct) model.
- Step 9: From the analysis result in Step 8, test cases are selected based on the mutation score achieved. This step removes the tests that are not contributing to a larger mutation score for the entire test suite.

The set of selected mutants used in MATS is created by adding mutations in the original model (e.g., changes in the rate of energy consumption for different components). These mutants can be created manually or automatically based on a predefined set of mutation operators [3]. These mutants should express naturally-occurring defects affecting energy consumption. For example, one can change the timing behaviour by adjusting the period and execution time conditions in the model (i.e., the period and execution time values).

These energy mutants are simulated using the same inputs generated by the reference model in each test case. The inputs of each test case are given as a predefined set of inputs for the mutated model. The mutated models with the predefined inputs are ready to be simulated. The result of these simulations represents the actual energy output signal which is stored and compared with the expected energy output signal generated initially from the original model. To evaluate the mutation result (i.e., energy oracle) each energy output is sampled using a user-defined granularity and then compared at each time point with the expected energy output to detect any significant differences larger than a user-defined delta value. MATS relies on the expertise of the engineers responsible for testing such systems. An experienced engineer should define what is an acceptable energy deviation. We note here that small deviations between the energy outputs are to be expected and the delta value can vary from one system to another.

IV. MATS Tool

The user interface of the MATS tool is shown in Figure 2. This interface contains several adjustable parameters. The user needs to choose multiple mutants based on the reference model. The query parameters are adjusted with the numericUp-Down controls from the Simulation Runs item, Period Count item, and Period Length item. These items are explained as follows:

- Simulation runs are used to define how many simulations will be generated on the reference model which corresponds to the number of generated test cases,
- Period Length is used to define how long each execution period is. This is represented in time units and is based on the system model's execution period to be simulated by UPPAAL SMC,
- Period Count stands for how many periods the simulation will execute. Both Period Length and Period Count are used to generate the Simulation Time,
- Sample Size allows the user to choose the granularity of the mutation analysis,
- Output Delta sets the delta limit for the output comparison based on the difference between the actual and mutated output signals at each time unit,
- The Input Query Parameters item allows the user to define which variables to be monitored as inputs,
- Output Query Parameter item allows the user to define which variables to be monitored as outputs. Both Sample Size and Output Delta values are used to calculate a quantitative measure of fault detection. A mutant is considered to be detected by a test suite if the values differ drastically

at certain time points. Sample Size decides how often this detection should be checked while the Output Delta defines how large the output difference can be.

In the MATS tool, we use mutation analysis [7], a method used to analyze tests by evaluating the ability of these tests to detect small changes in the software (e.g., to evaluate the effectiveness of a test suite). These small changes in the software (also called mutants) are manually or automatically created based on a set of mutation operators or manually injected faults. MATS detects if a test suite kills energy mutants if the energy signal diverges drastically at certain time points from the expected values (e.g., substantial energy deviations). If there is at least one energy value for which the distance is larger than the expected threshold, then we consider the mutant detected.

MATS is automatically generating simulation runs using UPPAAL SMC for a model described in TA extended with energy consumption. MATS is then transforming these simulations into actual test cases. A simulation run produced by the model checker for a given random run defines the set of inputs in time executed on the model, which in our case is considered the system model. Test cases are obtained by extracting from the test path the observable input values at any given time. The automatic test generation step in MATS is implemented using the UPPAAL SMC command line and extracts the parameter names and the points pairs representing the simulation trace as (x,y) points in a vector. MATS needs to sample this trace at predefined data points to retrieve all the needed information for test execution.

We mention that MATS does not automatically check if the used mutants are equivalent. This is a problem when using automatically generated mutants since these mutations keep the model semantics unchanged and thus cannot be detected by any test case. In our study, we used manual mutant equivalence checks, and we say that two models are equivalent if there is no input parameter for which the difference in energy consumption of the models exceeds some predefined threshold within some bounded time limit.

V. Experimental Results

To perform experiments, we use the MATS tool to generate test suites, measure the time needed to generate these suites and the mutation coverage reached with different parameters. For example, as shown in Figure 3, for a set of parameters (Simulation runs: 25, Simulation Time: 64, Sample Size: 0.050, Output delta: 4.0), two test suites will be automatically generated and suggested to the engineer. The test suites generated for the BBW system kill all mutants, which results in full mutation coverage[2]. In order to fine-tune these parameters and show the applicability of this approach, we repeated this process for different parameters. The reference model selected

[2]As the results show, energy mutants at the model level are not as easily killed. This can be explained by the fact that energy consumption needs to diverge drastically at multiple time points. This differs from studies using mutants created at the code level, which can be killed in some cases by even trivial test cases.

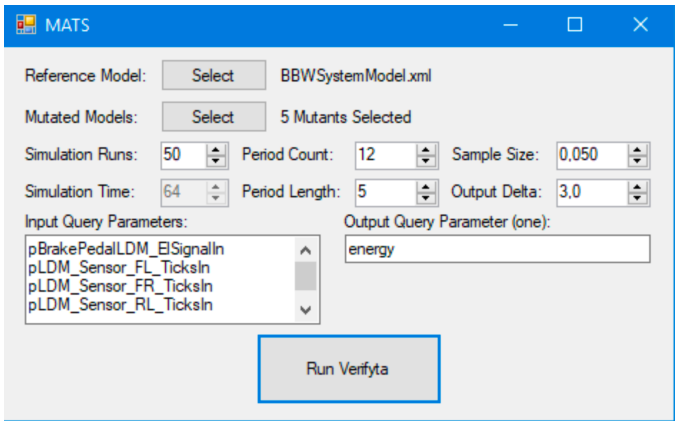


Fig. 2. User Interface of the MATS tool.

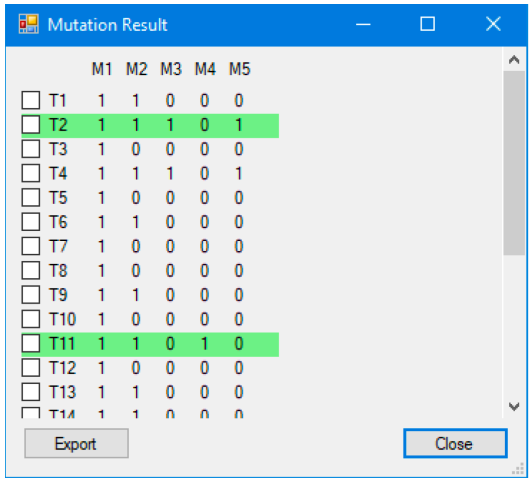
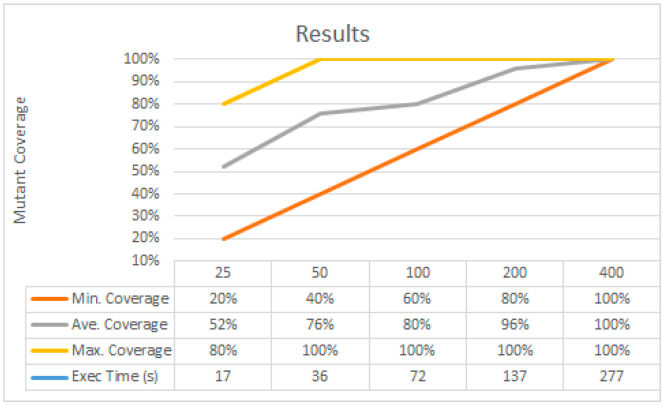


Fig. 3. Example of a test selection result using mutation analysis using the MATS tool.

in our case study is the BBW model, which contains a network of 50 timed automata divided into 25 pairs, 16 of which are computational blocks.

The BBW prototype is a braking system equipped with an anti-lock brake system (ABS) and is controlled electronically instead of mechanically. A sensor is attached to the brake pedal to read its position, which is used to compute the desired force applied to the brakes. With additional sensors to measure each wheel's speed, the ABS algorithm and the desired brake force, the actual brake torque is calculated and applied. This prototype system was originally described in EAST-ADL [3], an architectural language dedicated to automotive systems with support for resource annotations. A transformation from EAST-ADL to UPPAAL SMC was proposed by Marinescu et al. [2]. This model represents an industrial brake-by-wire system developed by industrial engineers. Five manually created mutants of the BBW model are used in this experiment. The BBW Prototype is a result of this transformation and is then manually extended with energy



	25	50	100	200	400
Min. Coverage	20%	40%	60%	80%	100%
Ave. Coverage	52%	76%	80%	96%	100%
Max. Coverage	80%	100%	100%	100%	100%
Exec Time (s)	17	36	72	137	277

Fig. 4. The test generation results on the BBW system; X-axis represents the number of tests generated by the MATS tool (i.e., 25, 50, 100, 200, 400).

information. The transformation generates two TAs for each component, one based on the component's interface and one based on the component's behaviour. For more details about this transformation, we refer the reader to the original paper [2].

The results of our measurements are shown in Figure 4. The result indicates that mutation coverage keeps increasing until it reaches 100% (when MATS is generating 400 test cases). Generating these 400 test cases takes 277 seconds on average when using the MATS tool on an Intel i5 series 4.5 GHz processor with 8GB of RAM. This shows that MATS is useful and efficient when using manually injected mutations. We note here that the generation time increases linearly with the number of simulations and mutants.

VI. Discussions and Conclusions

The obtained results could be useful for industrial engineers, test generation tool developers and researchers.

MATS does not rely on UPPAAL SMC's built-in simulation sampler. We found out that this sampler does not support resampling without simulating the model. To overcome this, we extract the raw data output and manipulate the data later using our own MATS tool.

In this paper, we have shown how UPPAAL SMC can be used for test generation and mutation analysis using energy consumption deviations at the model level. In addition, the MATS tool selects test suites contributing to the overall mutation score. Future work aims to apply this tool to a more industrial use case to expose its strengths as well as limitations both in terms of test efficiency and effectiveness.

VII. Acknowledgment

This work has received funding from H2020 under grant agreement No. 737494, from Vinnova through the SmartDelta projects and from KKS through ACICS project.

References

[1] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International journal on software tools for technology transfer*, 17:397–415, 2015.

[2] Raluca Marinescu, Eduard Enoiu, Cristina Seceleanu, and Daniel Sundmark. Automatic test generation for energy consumption of embedded systems modeled in east-adl. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 69–76. IEEE, 2017.

[3] Eduard Paul Enoiu and Cristina Seceleanu. Model testing of complex embedded systems using east-adl and energy-aware mutations. *Designs*, 4(1):5, 2020.

[4] Bernhard K Aichernig, Florian Lorber, and Dejan Ničković. Time for mutants—model-based mutation testing with timed automata. In *Tests and Proofs: 7th International Conference, TAP 2013, Budapest, Hungary, June 16-20, 2013. Proceedings 7*, pages 20–38. Springer, 2013.

[5] Florian Lorber, Kim G Larsen, and Brian Nielsen. Model-based mutation testing of real-time systems via model checking. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 59–68. IEEE, 2018.

[6] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinora, Italy, September 13-18, 2004, Revised Lectures*, pages 200–236, 2004.

[7] Allen T Acree, Timothy A Budd, Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. Mutation analysis. Technical report, Georgia Inst of Tech Atlanta School of Information And Computer Science, 1979.