24/1 (2026), 127–144
DOI: 10.5485/TMCS.2026.16815
tmcs@science.unideb.hu
https://ojs.lib.unideb.hu/tmcs

Teaching
Mathematics and
Computer Science

MTR framework for teaching model-based testing

GÁBOR ÁRPÁD NÉMETH and MÁTÉ ISTVÁN LUGOSI

Abstract. In the current article, it is presented how $Model \gg Test \gg Relax$ (MTR), a free and open-source, extended finite state machine model-based testing framework can be used for education purposes. With the education-related features of MTR – such as graph visualizations, subsequence creation, test suite export – the students are able to understand the concept behind model-based testing, the working of different model conversion and test generation algorithms. With project works, the students use the MTR framework for the automatic test design of a simplified, small scale real-world example. The framework also provides a simulation script for comparing the complexities and fault detection capabilities of different test generation algorithms.

Key words and phrases: model-based testing, finite state machines, extended finite state machines, conformance testing, test design, teaching.

MSC Subject Classification: 68M15.

Introduction

Testing is crucial in the software development process. As the complexity of software products continuously increases and release cycles become shorter and shorter, the probability of faults is also increasing. To solve this problem, the testing process must be automated. An approach is when the specification of the system is described with a formal model, and the test cases are derived from this model automatically according to some preset test criteria. This area of testing is called *model-based testing* (MBT).

To teach the MBT approach, the "Modelling and testing" course has been introduced in the ELTE Computer Science MSc program (Németh, 2020) that focuses on MBT using Finite State Machine (FSM) specifications, which models have been extensively used in different problem domains for decades (Holzmann 1990; Hercog, 2020; Ammann & Offut, 2016).

To illustrate the main concept behind FSM MBT, initially already existing MBT tools were used. Commercial tools – such as Conformiq Designer[1] and Reactis[2] – were simply too expensive to be selected, and also they are not open source, which limits their future extensibility. The open-source and free alternatives that are actively under development were GraphWalker (GW)[3], fMBT[4], and Modbat[5]. Although the Graphical User Interface (GUI) of GW Studio provides an easy first step to the MBT topic (Németh, 2020), its test generation is based mainly on random traversals of the specification, i.e., it lacks any efficient systematic test generation algorithms (Zafar et al., 2021). Also note that GW does not entirely follow the FSM formalism, which results in confusion in FSM MBT education. fMBT focuses on test generation from converted Extended FSMs. Its strategy relies on using random and other heuristics to fulfill a given coverage (such as permutations of consecutive elements), which is fine for some augmenting education purposes, but not to demonstrate given fault coverages. Also the visual representation features of fMBT are very limited: just the converted state/variable combinations of the specification can be visualized. Modbat can be used to test the application programming interface (API) of a software (Artho et al., 2015), but can not be used for GUI testing. Also, test cases can be generated only by some basic heuristic search, and this tool lacks any visualization features suitable for education. For the above reasons, a new MBT tool was developed.

In the current paper, it is shown how a new, free, open-source MBT framework called $Model \gg Test \gg Relax$ (MTR)[6] can be used in MSc education. Initially, MTR was created just for research purposes – to develop new systematic and heuristic test generation algorithms (Németh & Lugosi, 2021, 2024), to compare their complexities (Németh & Lugosi, 2024) and fault detection capabilities (Németh, 2025) with other solutions. Later, this framework has been

[1]Conformiq Designer, `https://www.conformiq.com/products/conformiq-designer-automated-test-design-embedded-systems`

[2]Reactis, `https://www.reactive-systems.com/products.msp`

[3]GraphWalker, `https://graphwalker.github.io/`

[4]fMBT, `https://github.com/intel/fMBT`

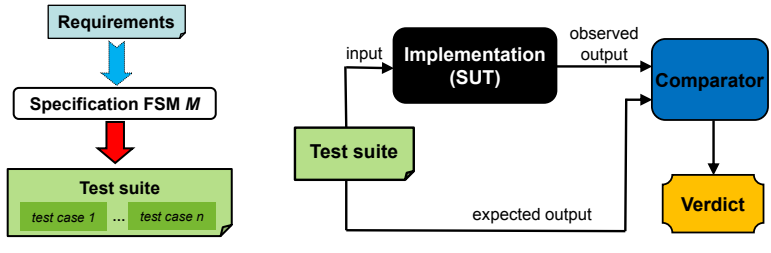
[5]Modbat, `https://gitlab.com/cartho/modbat`

[6]$Model \gg Test \gg Relax$, `https://modeltestrelax.org/`

extended with features aimed for education purposes and with a configuration profile optimized for that use case.

Finite State Machine model-based testing

A *Finite State Machine* (FSM) M is a quadruple $M = (I, O, S, T, s_0)$, where I, O, S and T are the finite and non-empty sets of *input symbols*, *output symbols*, *states* and *transitions* between states, respectively; s_0 denotes the *initial state*, from which M starts. Each transition $t \in T$ is a quadruple $t = (s_j, i, o, s_k)$, where $s_j \in S$ is the start state, $i \in I$ is an input symbol, $o \in O$ is an output symbol and $s_k \in S$ is the next state. An FSM can be represented with a directed labeled graph whose nodes and edges correspond to the states and transitions, respectively. Each edge is labeled with the input and the output, written as i/o, associated with the transition. The *Extended Finite State Machine* (EFSM) is an extension of the FSM formalism with variables and operations based on variable values. A more detailed description of these models, and their usual assumptions are discussed by Németh (2020).



(a) Test generation (b) Conformance testing

Figure 1. FSM model-based test generation and testing

In FSM model-based testing (MBT), an FSM model, denoted M, is created from the product's requirements. The *test cases* – consisting of input sequences and their expected output sequences – are generated from M according to some predefined criteria – see Figure 1(a). The collection of test cases is called *test suite*. To bridge the gap between the abstract model and the physical System Under Test (SUT), an *adaptation code* is required to implement M's transitions. This code allows abstract test suites to be converted into executable ones. Because the

SUT is treated as a *black box*, *conformance testing* checks if the *observed output sequence* of the implementation matches the *expected output sequence* derived from the specification – see Figure 1(b).

Model ≫ Test ≫ Relax framework

The high-level overview of the architecture of the *Model ≫ Test ≫ Relax* (MTR) MBT framework is presented in Figure 2.

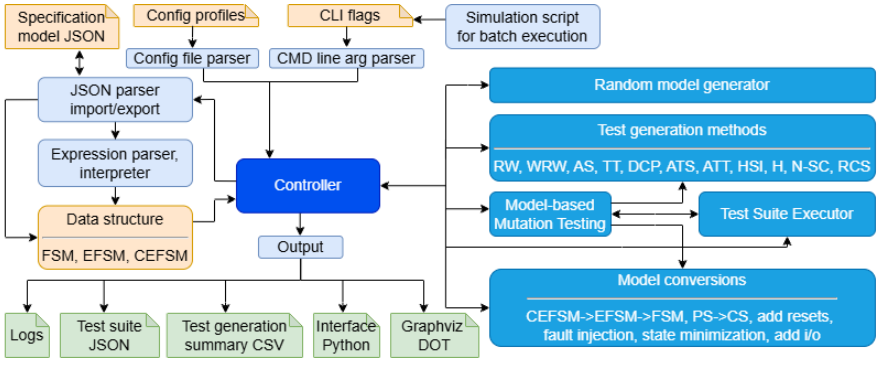


Figure 2. High-level overview of *Model ≫ Test ≫ Relax* R6

The framework is able to handle FSM, EFSM and CEFSM (Communicating EFSM) models and make conversions between these models. The tool can be used to generate test suites and provides an interface to evaluate results.

Three different configuration profiles are delivered with MTR that are optimized for testing, research and education purposes, called *DEFAULT*, *RESEARCH* and *EDUCATION*, respectively. The settings of the *EDUCATION* profile provides a more detailed logging and the generation of some augmenting data that can be used when one would like to understand the working of a given algorithm at the cost of a longer execution time compared to the other profiles. The *RESEARCH* profile is optimized for pure performance.

Model handling

MTR uses a JSON format to define specification models, which is an extension of the format used by GraphWalker (GW)[3]. Thus, the user can import models

edited in the GW Studio GUI. However, attention should be paid to the followings, as the model of GW does not entirely follow the (E)FSM formalism:

- The edges of a GW model do not have corresponding input and output symbols[7] which are required by MTR. If one adds this information to the edge name after a "|" character in the "*input/output*" form in GW Studio, it can be later converted and processed with MTR.
- In GW, the verification is done within the states, but this does not conform the (E)FSM formalism either, where the behaviour of the SUT can be observed through just its output.

In addition, MTR provides a wider range of model opportunities. Besides number (integer or float) and boolean variables – which can be used in GW as well – the user can also define string, enum and struct variables or ternaries. Also, more types of operators (which can be used in actions or guard conditions) are available compared to GW. Besides FSM/EFSM models, the user can define CEFSM models as well. Using CEFSMs, the specification can be described with a set of component EFSM machines, which communicate with each other through sending and receiving messages that can be associated to the transitions of component machines. Many example FSM, EFSM and CEFSM models and their descriptions are delivered with MTR[8].

Model conversions

MTR has a wide range of model conversion options, the most important ones from the point of view of MBT education are the following:

- *EFSM to FSM conversion:* For each state-variable value combination – which is reachable from the initial state of the EFSM – a distinct state is created. The conversion may result in the state explosion problem, but one can limit the range of variables[9]. With this feature, one can execute FSM test generation algorithms on the converted model, but the adaptation keywords need to be implemented only once for each EFSM transition.

[7]This information does not exist at model level, it can be handled at adaptation code level only.
[8]Example models are located in the `/sample_models` subfolder of MTR git repository and can be classified into three groups: *introduction examples* used in the "Modelling and testing" course, *application examples* which contain adaptation code besides the specification machine, and *test models* that are used in the automatic tests of MTR.
[9]These ranges can be set in the EFSM model file, using a similar approach as fMBT[9] does.

- *CEFSM to EFSM conversion:* From all possible combinations of states of the different CEFSM component machines, a new state is created in the converted EFSM model. The resulting EFSM can be further converted into an FSM.
- *Partially specified to completely specified conversion:* For each undefined state-input pair, a loop transition or a transition leading to a sink state without an output is added in one step.
- *Inject random faults* into the model to investigate the fault detection capabilities of different test suites.
- *State minimization* by merging equivalent states.

Test generation

MTR provides a wide range of test generation algorithms, see Table 1. A detailed overview of the available algorithms that compares their analytical and practical complexities is presented in (Németh & Lugosi, 2024).

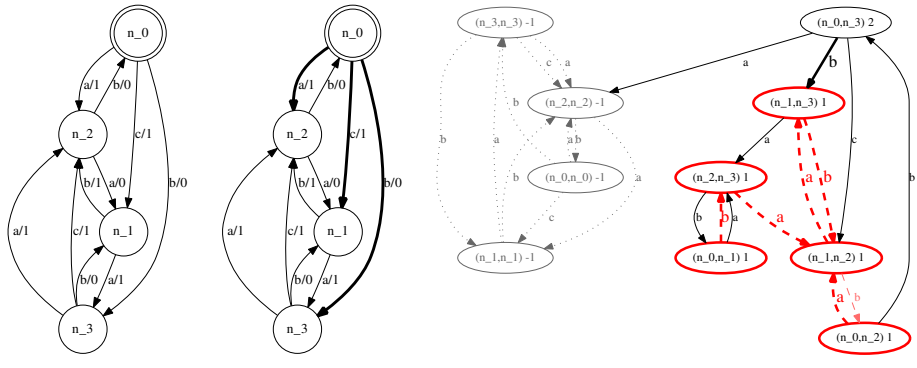
Category	**Algorithm**	**Reference**	**Education-related features**
Random	Random Walk (R)		GWe
	Weighted Random Walk (WR)		GWe
Heuristic algorithms for a given coverage criteria	All-State (AS)	(Gutin et al., 2002)	GWe
	All-Transition-State (ATS)	algo: (Németh & Lugosi, 2021), criteria: (Forgács & Kovács, 2019	Gv+SS+GWe
	All-Transition-Transition (ATT)	(Forgács & Kovács, 2019)	SS+GWe
	N-Switch Coverage (N-SC)	algo: (Németh & Lugosi, 2024), criteria: (Chow, 1978)	Map+GWe
Optimal tour for transition coverage	Transition Tour (TT)	(Naito & Tsunoyama, 1981)	Gv+GWe
	Directed Chinese Postman (DCP)	(Edmonds & Johnson, 1973)	Gv+GWe
Structured algorithms with state verification	Harmonized State Identifiers (HSI)	(Luo et al., 1995)	Gv+SS+GWe
	H-method (H)	(Dorofeeva et al., 2005)	Gv+SS+GWe
	Reset Checking Sequence (RCS)		GWe
Upper layer above all algorithms	Model-based mutation testing	(Németh, 2025)	Map

Table 1. Test generation algorithms available in MTR

The following supplementary education-related features can be used when one would like to understand how a given test generation algorithm works:

- Graphviz graph visualizations (Gv);
- Subsequences of the test suite (SS);
- Data map structures for given coverage (Map);
- Export the generated test suite into a format that can be played back step-by-step with GraphWalker Studio in a GUI (GWe).

Generating Graphviz graph files and subsequences



(a) FSM M (b) State cov. set Q (c) Auxiliary state pair graph of M

Figure 3. Graphviz file generation examples

Consider the FSM M presented in Figure 3(a), where the double circle denotes the initial state[10]. Generate a test suite (`-o t`[11]) with the HSI algorithm (`-m HSI`) with graph visualization (`--graphviz`) and subsequences (`--subsequences`) options for this model using the following command:

```
MTR -m hsi -f sample_models/introduction/h_1.json --graphviz --subsequences
```

Briefly, the HSI test generation works as follows. It contains two parts:

- A *state cover set* $Q = \{q_0, \ldots, q_{n-1}\}$ responsible for reaching all states of M; the problem is reduced to creating a spanning tree from the initial state s_0.
- A *separating family of sequences* of Z is used to verify the end states. The set Z is a collection of sets $Z_j, j = 0, \ldots, n-1$ of sequences (one set for each state) where for every non-identical pair of states s_j, s_l, there exists a separating sequence that provides a different output for the two states.

[10]This example is delivered in `sample_models/introduction/h_1.json` of MTR git repository.
[11]Note that this flag can be omitted as test generation is the default operation mode of MTR.

Built on the above parts, the algorithm has two stages. The first one identifies all states of the FSM, and the second one checks all remaining transitions.

As MTR shows, the overall length of the generated test suite is 46.

Subsequences: Besides the input and output lists of the entire test suite, the following subsequences are added to the test suite:

```
"q_state_cover_set_input_list": {
 "q_n_0": [],
 "q_n_1": ["c"],
 "q_n_2": ["a"],
 "q_n_3": ["b"]},
"z_separating_family_of_sequences_input_list": {
 "z_n_0": {"z(n_0,n_1)": ["b"], "z(n_0,n_2)": ["a"], "z(n_0,n_3)": ["b", "b"]},
 "z_n_1": {"z(n_0,n_1)": ["b"], "z(n_1,n_2)": ["a"], "z(n_1,n_3)": ["b"]},
 "z_n_2": {"z(n_0,n_2)": ["a"], "z(n_1,n_2)": ["a"], "z(n_2,n_3)": ["a"]},
 "z_n_3": {"z(n_0,n_3)": ["b", "b"], "z(n_1,n_3)": ["b"], "z(n_2,n_3)": ["a"]}},
"transition_testing_input_list": {
 "transtest_n_1": [["a"], ["b"]],
 "transtest_n_2": [["b"], ["a"]],
 "transtest_n_3": [["c"], ["a"], ["b"]]}
```

Generated Graphviz files: The resulting spanning tree – denoted by bold edges – representing state cover set Q is shown in Figure 3(b). The state pair graph representing separating family of sequences Z is presented in Figure 3(c). Here, red bold ellipses and red bold dashed edges with bigger label denote separating state pairs and separating inputs selected by the algorithm[12], and black bold edges with bigger label represent spanning forest edges pointing to separating state pairs. The number after the state pair name shows the length of the given separating sequence. The testing tree is shown in Figure 4. Here, the double red circle, simple red circles and orange circles represent the initial state, other states reached at the state verification stage by a q_k sequence and states reached at the transition testing stage, respectively. The entire test suite builds up by traversing each possible path from the initial state to the leaves of the spanning tree.

To see how the H-method is an improvement over HSI test generation, execute the following command using the H-method on the same model:

```
MTR -m h -f sample_models/introduction/h_1.json --graphviz
```

The overall length of the resulting test suite is 40, which is shorter than the one created by HSI. The testing tree of the H-method is shown in Figure 5. The H-method prioritizes the minimization of the branching extension of the testing tree (that would result in more test sequences from the initial state) instead of selecting the shortest sequence (i.e., to minimize the depth of a given branch). Thus, the

[12]Light red dashed edges with smaller label show another possible separating inputs that were not selected by the algorithm.

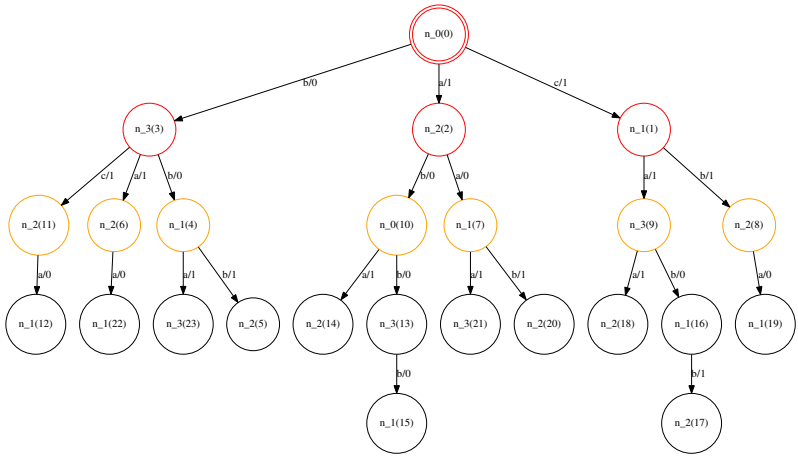


Figure 4. Testing tree of M generated by the HSI-method

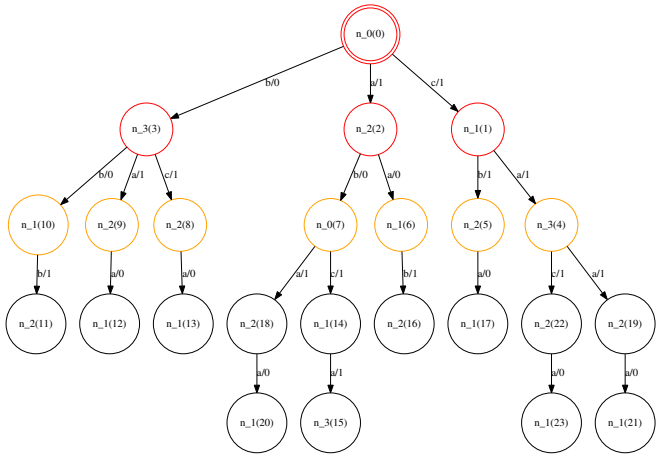


Figure 5. Testing tree of M generated by the H-method

testing tree contains fewer branches (i.e., fewer test sequences) compared to the one generated by HSI.

Note that as shown in Table 1, graph visualizations and subsequences can be used for other test generation algorithms as well, and for N-SC and model-based mutation testing, coverage maps can also be exported – for details, see the user guide[16].

Export test suite into GraphWalker Studio

One can also export the generated test suite into the JSON model file, to be executable in GW[13]. Note that special model elements, that are not handled by GW – such as reset symbols that take the machine from any state back to the initial state – are also automatically added explicitly to the model by MTR.

Apply HSI test generation with `--predefined_path`, which adds the transition list of the test suite to the model:

```
MTR -m hsi -f sample_models/introduction/h_1.json --predefined_path
```

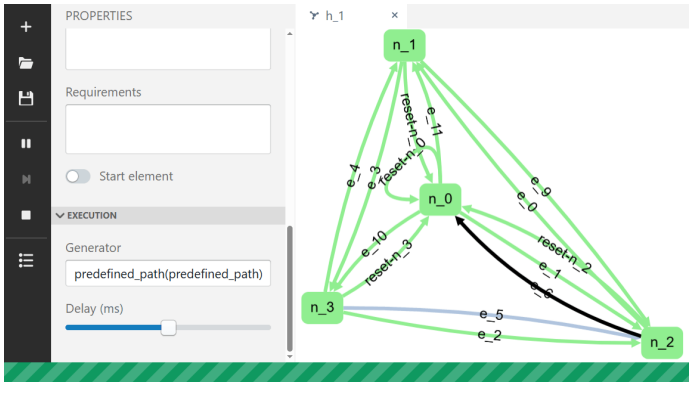


Figure 6. Executing a test suite generated by MTR in GW Studio

Open the exported model with the test suite in GW Studio, and press the "play" button at the right pane to check the execution of the test suite on the GUI – see Figure 6 [14]. Transitions 'already executed', 'currently under execution' and 'not yet visited' are displayed by green, black and blue colours, respectively.

Investigate the complexity of different test generation algorithms

The simulation script can be used to compare the test generation complexity, the size of the resulting test suite and the fault detection complexity of different test generation algorithms.

Go to the `simulations/scripts` folder of simulation script. Generate completely specified, reduced models with increasing number of states, from 5 to 200 with a step size of 5, density 10, output symbol count 10 and with reliable reset:

[13]This feature from GW side was implemented by us and has been introduced in GW 4.3.2.
[14]GW Studio displays the name of the transitions instead of their input/output symbols.

```
python3 run_sims.py --model_gen s 5 200 5 d 10 o 10 spanning c reduced reliable
```

The generated models are collected into a folder following the `models_s_5_200_5_d_10_o_10_st_cs_reduced_r_<hash>` name format.

Rename the folder of generated models to `Scenario1`. Apply, TT, ATS, HSI and H test generation on these models using *RESEARCH* profile:

```
python3 run_sims.py --batch_exec Scenario1/ TT ATS HSI H --profile RESEARCH
```

For the above command, summary result CSV files and an xlsx file have been created. Rename the folder of these files into `Results`. Create a diagram that shows the size of the generated test suites in function of number of states (using data from `state_count` and `test_sequence_length` columns of CSV files):

```
python3 run_sims.py --plotly state_count test_sequence_length Results/tt_result.csv
Results/ats_result.csv Results/hsi_result.csv Results/h_result.csv
```

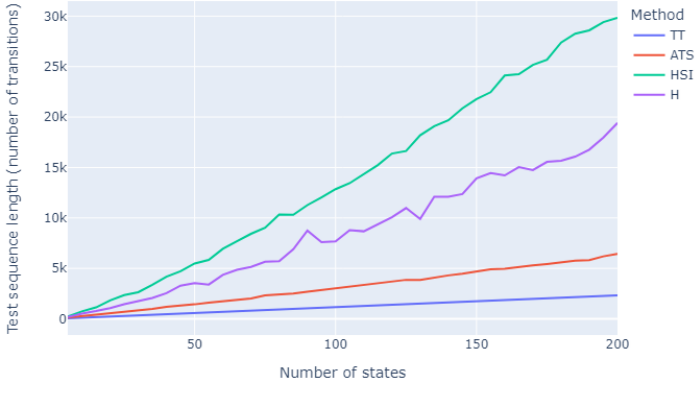


Figure 7. The results of simulation script for batch execution

See the resulting diagram in Figure 7. Besides the output data, the parameters of the model, the settings of the algorithms and the used MTR version[15] are also logged in the summary xlsx/CSV files, which makes the results reproducible.

Note that in this section only a small demonstration example was presented on the simulation script, the test generation and execution complexity of different algorithms, and the fault detection capability of the resulting test suites are investigated in detail in (Németh & Lugosi, 2024) and (Németh, 2025), respectively.

[15]The version number of MTR is updated automatically after the *master* branch is updated.

Test projects and development with Model ≫ Test ≫ Relax

SUT	Domain	Model	Adaptation code	MTR example
The SUT is a webpage or an open-source application or library:				
Discord	Instant messaging and VoIP social platform	EFSM	Python, API, GUI: Selenium	-
Elvira	Train ticket web app.	FSM	Python, GUI: Selenium	-
Google Keep	Note taking app.	FSM	Python, API	-
Google Calendar	Calendar web app.	FSM	Python, Google Cal. API	-
Git	Version control system	FSM	Python, CLI	Yes[17]
Instagram	Photo & video sharing social platform	FSM	Java, GUI: Selenium	-
Jira	Issue tracking system	EFSM	Python, API	-
Kubernetes	Container orchestration platform	FSM	Java, CLI	-
Neovim	Text editor (Unix)	FSM	Python, API	Yes[17]
OpenIddict	OAuth2.0/OpenID protocol library	EFSM	C++, API	Yes[17]
Ory Hydra	OAuth2.0/OpenID Connect server	FSM	C#, API	Yes[17]
OrangeHRM	Human resources management system	EFSM	Python, GUI: Selenium	-
Reddit	Social media platform	FSM	Python, Reddit API	-
SOCKS5	Proxy server	FSM	Python, API	-
Spotify	Audio streaming service	FSM	Python, Spotify Web API (Node.js)	-
Steam	Online marketplace	FSM	Python, GUI: Selenium	-
Telegram	Safe chat app.	EFSM	C#, Telegram API	-
Twitter	Social media & microblogging platform (now "X")	FSM	Java, Twitter API v2	-
Unity Asset Store	Online marketplace	EFSM	TypeScript, GUI: Selenium	-
Wikipedia	Online encyclopedia	EFSM	Python, GUI: Selenium	-
XSync 3.1	X Synchronization Extension Library (Linux)	EFSM	C++, xcbX protocol lib.	-
The SUT is an application which was created for BSc Thesis or for a programming course:				
AlcoholLimit	Android app.	FSM	Python, Appium	-
Animal Shelter	Web app.	FSM	JavaScript, GUI: Playwright	-
ATC	Simulator app. of Automatic Transmission Controller	EFSM	Java, GUI: AssertJSwing	-
Betting odds	Web app. for betting odds analyses	FSM	Java, GUI: Selenium, API: Postman	-
Cookie Shop	Webshop/backend	FSM	Python, API	-
Dungeon crawler	Console game	EFSM	C++ & python, CLI	-
ELTELive	Course streaming website	FSM	Python, API	Yes[17]
ELTE Schedule	Schedule planner for courses	FSM	TypeScript, GUI:Playwright	-
Farmer2Customer	Online marketplace	EFSM	Python, API	-
Food ordering	Web app. for food ordering	EFSM	C#, GUI: Playwright	-
Job Hunting	Web app. for job searching	EFSM	JavaScript, GUI: Playwright	-
MadTycoon	Tycoon simulator game	EFSM	Java, GUI: intelIJ	-
MisEnapló	Web app. for parishes	FSM	JavaScript, GUI: Selenium	-
PriceWatcher	Shopping assistant web app.	FSM	C#, GUI: Selenium	-
Train ticket	Train ticket web app.	FSM	C#, GUI: WinAppDriver	-
Webshop	ASP.NET webshop app.	FSM	Python, GUI: Selenium	Yes[17]

Table 2. MBT project works that used Model ≫ Test ≫ Relax

During the "Modelling and testing" course, students create MBT project works to test a real application. Although they can select the tool for their works freely, most of them select GW for editing the model in its GUI, then switch to

MTR for the rest of their work (model conversions, test generation and execution). The reasons for this are (1) the problems of the free MBT tools (mentioned in the Introduction), and the facts that (2) most of the algorithms presented in the course are implemented in MTR, (3) MTR is well documented[16], (4) and we provide a support for MTR. See the list of project works that used MTR in Table 2. The SUT can be either a website or an application available online or an application which was previously created for a BSc Thesis or for another course. Note that the model and the adaptation code of some of these testing projects are also delivered with MTR[17], which can be used as a starting point when one would like to create an own test project using MTR.

Ory Hydra example project

From the MTR projects presented in Table 2, Ory Hydra is selected to briefly demonstrate how an actual MBT project works. This application example – as all other MBT projects – contains the following three main parts: (1) a System Under Test (SUT), (2) a specification model, (3) an adaptation code.

SUT: Ory Hydra is an OpenID certified OAuth 2.0 (RFC 6749, 2012) server and OpenID Connect[18] provider which can be integrated into web applications to conduct different OAuth 2.0 flows. The SUT is the 2.2.0 version of Ory Hydra[19].

Specification model: The FSM specification model *Hydra_Client.json*[20] created for the SUT (shown in Figure 8 [14]) covers the following functionalities[21]:

- Register a new client / delete client registration;
- Login with authentication:
  - Initiate the authorization code flow and return a login challenge,
  - Return a login challenge's metadata based on a valid login challenge ID,
  - Accept/reject login requests for valid login challenge IDs,
  - Return challenge's metadata based on a valid consent challenge ID,
  - Accept/reject a valid consent challenge,
  - Exchange a valid authorization code for a valid access token;

[16]MTR User Guide can be found in `/docs/user_guide.md` path in MTR git repository.
[17]Application examples can be found in `/sample_models/applications` folder of MTR git repo.
[18]OpenID Connect, `https://openid.net/developers/how-connect-works/`
[19]Ory Hydra v2.2.0, `https://github.com/ory/hydra/tree/v2.2.0`
[20]Located in `/sample_models/applications/OryHydra/Hydra_Client.json` of MTR git repo.
[21]The detailed call flows are shown in `/sample_models/applications/OryHydra/OryHydra.md`

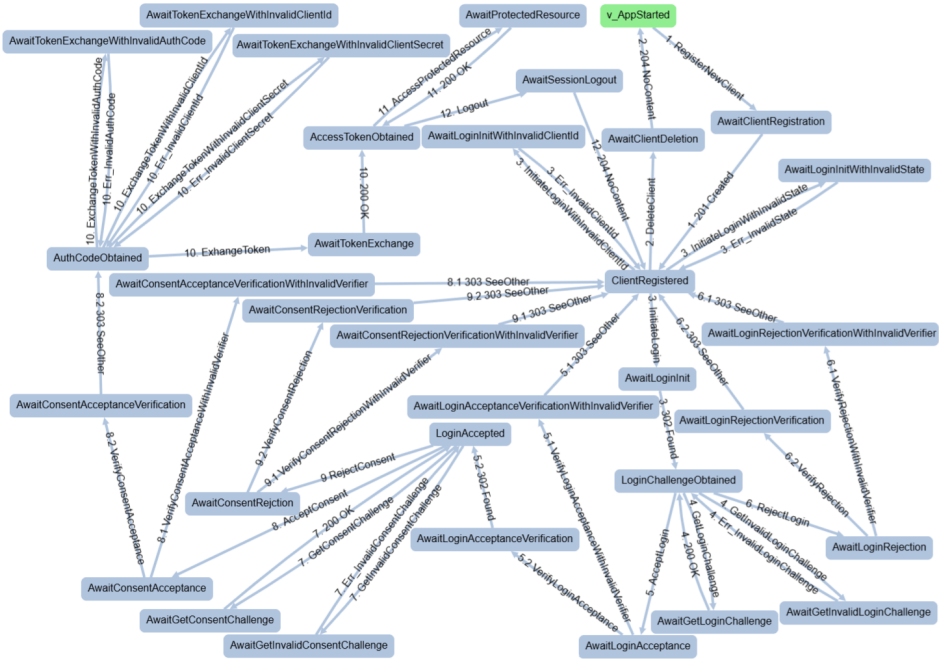


Figure 8. Specification FSM for the Ory Hydra example in GW Studio

- Access a protected resource using a valid access token;
- Suspend and active session.

The adaptation code: The adaptation code is written in C# and tests the SUT through an API. It consists of two projects:

- *Hydra*, which contains a single endpoint that acts as the protected resource. When the protected resource is accessed, the "Authorization" header is extracted, and the access token within is validated. It has the following files:
  - *Program.cs*, which is the entry point of the program. This configures the necessary settings to grant the program access to the protected resource.
  - *Controllers/HydraController.cs*, which contains the endpoint for the protected resource and the JWT (JSON Web Token) validation logic.
  - *Enums/CookieType.cs*, which contains the different types of cookies (login, session, consent) managed by Ory Hydra in the different scenarios.

- *Hydra_Test*, which contains the test programs implementation that executes the test suites generated by MTR. It contains the following files:
  - *Tester.cs*, which contains all the necessary variables, the parsing logic, and the API testing logic to perform the tests.
  - *Test_Sequences* folder, which contains the test suites generated by MTR.

Test generation: Generate test suites with MTR from the specification model using the AS, TT, ATS0, ATT and 2-SC algorithms, respectively:

```
MTR -m AS -f sample_models/applications/OryHydra/Hydra_Client.json
MTR -m TT -f sample_models/applications/OryHydra/Hydra_Client.json
MTR -m ATS --ats_depth 0 -f sample_models/applications/OryHydra/Hydra_Client.json
MTR -m ATT -f sample_models/applications/OryHydra/Hydra_Client.json
MTR -m NS --ns_value=2 -f sample_models/applications/OryHydra/Hydra_Client.json
```

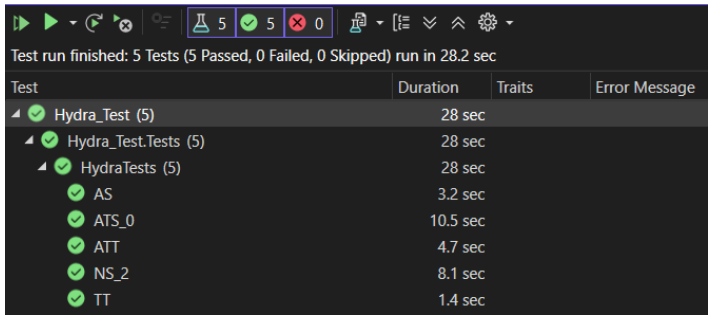


Figure 9. Running the tests generated with MTR in Visual Studio

Test Execution: Execute the test suites in Visual Studio: right-click on *Hydra_Test*, then select *Run tests*. The results show that all tests passed, see Figure 9. One can also apply the following command (in the *OryHydra* folder, where the *Hydra.sln* file is located) to run these test suites in the command line:

```
dotnet test --logger "console;verbosity=detailed"
```

Involve students in framework development

The frequent use of MTR for MBT project works provides us an opportunity to get feedback twice a year about the usefulness of new MTR features, identify possible bugs in MTR or problems related to the User Guide. This information is then used to improve MTR in the next release. Some of the students participating in the course also apply later to a MBT topic to write their MSc Thesis. They implement their corresponding algorithm in MTR and use the simulation script

to compare its performance to already existing solutions. To help their productive work from the first steps, the setup of the repository and the environment, our working process, the concept behind different parts of MTR and our coding conventions are described in our notes for contributors documentation[22], and the source code is commented using doxygen.

Conclusion

In this paper, it is discussed how $Model \gg Test \gg Relax$, an open-source model-based testing framework can be used for education purposes in the "Modelling and testing" course of our MSc program. The wide range of model conversion, test generation and education-related augmenting features (such as graph visualizations, subsequences, export test suites), the delivered sample models and test projects help students to understand the concept behind model-based testing, shorten the learning curve of this topic, and to create their own model-based testing project work during the term. The frequent use of the $Model \gg Test \gg Relax$ framework by our students results in feedback that can be used to improve the tool. Some of the students participating in the course also write their MSc Thesis later using this framework.

Acknowledgements

The authors would like to thank the students who participated in the implementation of the following parts of the MTR framework: Tódor Dávid Nyeste for the H and HSI test generation, Bálint Miksa Rumpler for the N-SC test generation and model-based mutation testing, Zsolt Csáky for EFSM handling, and EFSM → FSM transformation, Ádám Bába for CEFSM handling, CEFSM → EFSM transformation and reset handling, Fruzsina Mária Habzda for the refactoring of Graphviz file generation. Zsolt Csáky also implemented a feature from GraphWalker side that allows our test suites to be imported into this tool. Levente Székely implemented the Ory Hydra example project. Last, but not least, the authors would like to thank the students giving feedback about the MTR tool while using it to solve their project work.

[22]See `docs/notes_for_contributors/notes_for_contributors.md` in MTR git repository.

References

Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/9781316771273

Artho, C., Seidl, M., Gros, Q., Choi, E.-H., Kitamura, T., Mori, A., Ramler, R., & Yamagata, Y. (2015). Model-based testing of stateful APIs with Modbat. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering* (ASE 2015) (pp. 858–863). IEE. https://doi.org/10.1109/ASE.2015.95

Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, *4*(3), 178–187. https://doi.org/10.1109/TSE.1978.231496

Dorofeeva, R., El-Fakih, K., & Yevtushenko, N. (2005). An improved conformance testing method. In F. Wang (Ed.), *Formal Techniques for Networked and Distributed Systems – FORTE 2005* (Lecture Notes in Computer Science, Vol. 3731) (pp. 204–218). Springer. https://doi.org/10.1007/11562436_16

Edmonds, J., & Johnson, E. L. (1973). Matching, Euler tours and the Chinese postman. *Mathematical Programming*, *5*(1), 88–124. https://doi.org/10.1007/BF01580113

Forgács, I., & Kovács, A. (2019). *Practical test design*. BCS, The Chartered Institute for IT.

Gutin, G., Yeo, A., & Zverovich, A. (2002). Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics*, *117*(1-3), 81–86. https://doi.org/10.1016/S0166-218X(01)00195-0

Hercog, D. (2020). Protocol specification and design. In *Communication protocols*. Springer. https://doi.org/10.1007/978-3-030-50405-2_2

Holzmann, G. J. (1990). *Design and validation of computer protocols*. Prentice-Hall.

Luo, G., Petrenko, A., & v. Bochmann, G. (1995). Selecting test sequences for partially-specified nondeterministic finite state machines. In T. Mizuno, T. Higashino, & N. Shiratori (Eds.), *Proceedings of the IFIP WG6.1 7th International Workshop on Protocol Test systems VI* (pp. 95–110). Springer. https://doi.org/10.1007/978-0-387-34883-4_6

Naito, S., & Tsunoyama, M. (1981). Fault detection for sequential machines by transition tours. In *Proceedings of the 11th IEEE Fault-Tolerant Computing Conference (FTCS 1981)* (pp. 238–243). IEEE Computer Society Press.

Németh, G. Á. (2020). Teaching model-based testing. In *Teaching Mathematics and Computer Science*, *18*(1), 1–17. https://doi.org/10.5485/TMCS.2020.0469

Németh, G. Á. (2025). Model-based mutation testing for Finite State Machine specifications with MTR. *Infocommunications Journal*, *17*(3), 84–91. https://doi.org/10.36244/ICJ.2025.3.10

Németh, G. Á, & Lugosi, M. I. (2021). Test generation algorithm for the All-Transition-State criteria of Finite State Machines. *Infocommunications Journal*, *13*(3), 56–65. https://doi.org/10.36244/ICJ.2021.3.6

Németh, G. Á, & Lugosi, M. I. (2024). MTR Model-Based Testing Framework. *Infocommunications Journal*, *16*(2), 11–18. https://doi.org/10.36244/ICJ.2024.2.2

RFC 6749: The OAuth 2.0 Authorization Framework (2012). Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/html/rfc6749

Zafar, M. N., Afzal, W., Enoiu, E., Stratis, A., Arrieta, A., & Sagardui, G. (2021). Model-Based Testing in practice: an industrial case study using GraphWalker. In *Proceedings of the 14th Innovations in Software Engineering Conference (ISEC 2021)* (Art. ID. 5, 11 pp.). https://doi.org/10.1145/3452383.3452388

GÁBOR ÁRPÁD NÉMETH
DEPARTMENT OF COMPUTER ALGEBRA
FACULTY OF INFORMATICS, EÖTVÖS LORÁND UNIVERSITY
H-1117 BUDAPEST, PÁZMÁNY PÉTER SÉTÁNY 1/C., HUNGARY

E-mail: nga@inf.elte.hu

MÁTÉ ISTVÁN LUGOSI
FREELANCER, HUNGARY

E-mail: mate.lugosi@gmail.com

(Received February, 2026)