Vrije Universiteit Amsterdam

Bachelor Project

An Experience Report on Model Learning

Author: Gideon Roose (2645786)

Supervisor: Prof. dr. Wan Fokkink
2nd reader: dr. Jörg Endrullis

July 2022

Abstract

This paper dives deeper in how model learning can be used to learn automata for real-world systems. Most of the research done on model learning is theoretical and the tools are often not fit to be used in real systems. After describing how model learning works, we do some experiments in a tool called Tomte: one theoretical experiment to see how the tool uses certain input files to generate models and one experiment in which we try to construct a SUT for an external API, to get an understanding of the difficulties of applying model learning to real-world systems.

1

Introduction

Imagine you just bought your first smartwatch. How would you explore all its functionalities? To get to know all the features of the watch, most people would not read the manual from front to back. A more likely scenario is that you start pressing buttons and opening apps and menus to observe what happens. Then, by trial and error, you start to understand how it works.

This is the basic principle of model learning. To put it more formally, model learning is a method to construct a state-diagram model of a software or hardware system from which the exact behaviour is unknown (a black-box system) (1). It is also known as automata learning and there are two different types of learning algorithms: passive and active learning algorithms.

1.1 Passive automata learning

Algorithms based on passive automata learning attempt to construct a model by using a set of traces of the program. These traces are often collected from actual usage of a system. For example, take a device with three buttons: Button A, Button B and Button C. Each button outputs something different. The set of traces used for learning the systems could then be as follows:

- Press "Button A" -> Output: "a"
- Press "Button B" -> Output: "b"
- Press "Button C" -> Output: "c"

From these traces, a model can be constructed. There is, however, a fundamental flaw in this approach. What if it is also possible to press multiple buttons in sequence? Then

pressing "Button A" followed by "Button B" might result in the output "ab". Perhaps pressing "Button C" three times turns the system off. This basic example illustrates that with passive model learning it is extremely hard to be sure the constructed model is accurate, especially when the system increases in complexity. Furthermore, since the set of traces might not paint a complete picture of how the system works, it is also possible that one set of traces can result in multiple valid models, making the result ambiguous (2).

1.2 Active automata learning

Active model learning, on the other hand, is generally more thorough. Algorithms based on this approach learn by actively doing experiments on the system, instead of using a predefined set of traces like passive learning algorithms. Most active learning algorithms used today are based on Angluin's approach (3) of using a *minimally adequate teacher* (MAT). The teacher provides information about the *system under testing* (SUT) to the learner (the learning algorithm). The learner can use two types of queries that the teacher can answer: membership and equivalence queries. Membership queries are composed of a sequence of input symbols (2). When the learner provides such a query to the teacher, the teacher will respond with output from the system. By sending many of these membership queries to the teacher and observing the output returned by the teacher, the learner can construct a hypothesized model of how the system works. Then the learner can check if the model is accurate by sending an equivalence query, followed by checking whether the model is equal to the teacher model. If it is equivalent, the process is finished. If it is not equivalent, the teacher will return a counter-example that illustrates how the learned model differs from the teacher model. Based on this counter-example, the learner can send new membership queries in order to construct a new model. The remainder of this report will focus on active model learning.

As mentioned in (4), model learning has multiple realistic useful applications, like regression testing, replacing manual testing by model-based testing, producing models of standardized protocols, or analyzing whether an existing system is vulnerable to attacks. However, a problem with active automata learning is that it is quite difficult to implement it into your system. Most of the time, a solution is developed specifically for a certain system, meaning no one can easily reuse that implementation. Luckily, some tools aim to automate most of the work needed in automata learning. The question is, though, how easy it actually is to use these tools in a real-world environment.

2

Model Learning

To get a full understanding of what model learning is, it is useful to know which models it is generally used on, what algorithm most currently used learning algorithms are based on, and what kind of tools are available to make it easier to get started with model learning.

2.1 Target models

There exist various models for which automata learning algorithms are available. However, not all models can be learned with the currently existing algorithms. The models for which the largest number of algorithms are available are deterministic finite automata (DFA) and Mealy machines. However, in recent years, some progressions have been made for other models as well; one of those is the Register Automaton [1].

2.1.1 Deterministic Finite Automata (DFA)

The first model learning algorithms were designed for inferring regular languages, described as deterministic finite automata (5). Therefore, a short definition of a DFA is given below.

Definition 1 *A Deterministic Finite Automaton is a tuple* $(Q, \Sigma, \delta, q_0, F)$ *consisting of:*

- *A finite set* Q *of states;*
- *A finite input alphabet* Σ*;*
- *A transition function* $\delta : Q \times \Sigma \to Q$*;*
- *A starting state* $q_0 \in Q$*; and*
- *A set* $F \subseteq Q$ *of final states.*

[1] `https://wcventure.github.io/Active-Automata-Learning/`

A limitation to DFAs in the real world is that they do not produce output for every input. DFAs only accept certain sequences of input symbols (sequences that start in starting state q_0 and end in F), which means that for any input sequence, a DFA can only provide a single output, namely whether the input is valid or invalid. This, however, is often not enough to represent a real-world system.

2.1.2 Mealy machines

Most model learning algorithms used today have been improved so that they can support Mealy machines. A Mealy machine is a finite-state machine and its output is determined by its current state and current inputs.

Definition 2 *A Mealy machine is a tuple* $(S, S_0, \Sigma, \Lambda, T, G)$ *consisting of:*

- *A finite set* S *of states;*
- *A start state (also called the initial state)* S_0 *which is an element of* S*;*
- *A finite input alphabet* Σ*;*
- *A finite output alphabet* Λ*;*
- *A transition function* $T : S \times \Sigma \to S$*; and*
- *An output function* $G : S \times \Sigma \to \Lambda$*.*

The definition of a Mealy machine is quite similar to the definition of a DFA, but there are some slight differences. Note how Mealy machines have an output function G, whereas DFAs have a set F of final states. This results in an advantage of being able to learn a Mealy machine over simply a DFA as Mealy machines can produce output for every input symbol. Real-world systems can often be better represented by a Mealy machine than a DFA, but since this is not always the case, it is still useful that some algorithms work for DFAs just like some work for Mealy machines.

2.1.3 Register Automata (RA)

To understand what register automata are, one must first be aware of the differences between extended finite-state machines (EFSM) and finite-state machines (FSM). ESFMs are FSMs with the addition of state variables and several functions [1]:

[1] `https://automata.cs.ru.nl/Syntax/EFSM`

- Enabling functions are an expression over the input parameters and state variables to decide whether a transition is enabled;
- Update functions update the state variables when a transition is taken; and
- Output functions calculate the new output parameters when a transition is taken.

Definition 3 *A Register Automaton (RA) is a tuple* $(I, O, V, L, l_0, \Gamma)$ *consisting of:*

- *A set* I *of input symbols;*
- *A set* O *of output symbols;*
- *A finite set* V *of state variables, and we assume two special variables in and out not contained in* V *and write* $V_{i/o}$ *for the set* $V \cup \{in, out\}$*;*
- *A finite set* L *of locations;*
- *An initial location* $l_0 \in L$*; and*
- $\Gamma \subseteq L \times I \times \Phi(V_{i/o}) \times (V \to V_{i/o}) \times O \times L$ *is a finite set of transitions. For each transition* $\langle l, i, g, \varrho, o, l' \rangle$*, we refer to* l *as the source,* i *as the input symbol,* g *as the guard,* ϱ *as the update,* o *as the output symbol and* l' *as the target. We require that out does not occur negatively in the guard, meaning it may not occur in a subformula of the form* $x \neq y$*.*

Register Automata are Mealy extended finite-state machines (Mealy EFSM, also known as scalarset Mealy machines) and they place some restrictions on the extra functions of the EFSM; the update function can only assign a register (state variable) with a register or an input parameter (meaning no operations are allowed on data) and the enabling function is limited to a boolean expression of the form $true$, $false$, $x = y$ or $x \neq y$ where x and y are either registers or input parameters.

2.2 The L* algorithm

When Angluin designed the MAT model, she also developed the L* algorithm for learning DFAs (3). Interestingly, even though faster algorithms exist these days (than L* which runs in polynomial time), most still follow Angluin's MAT model (which she proposed in 1987). The L* algorithm works by following the MAT model as explained in section 1.2 and using an observation table to keep track of all the retrieved information the learner has gotten from the teacher.

Definition 4 *An observation table is a triple (S, E, T), consisting of:*

- *A nonempty finite prefix-closed set S of strings (prefix-closed means that every prefix of every member of the set is also a member of the set and the same goes for suffix-closed);*
- *A nonempty finite suffix-closed set E of strings; and*
- *A finite function T mapping $((S \cup S \times A) \times E)$ to $\{0, 1\}$, where A is a fixed known finite alphabet.*

To make it more clear what an observation table looks like, you can imagine it as a matrix where the rows are labelled by elements of $(S \cup S \times A)$, the columns are labelled by elements of E, and the entry at row s and column e is equal to $T(s \times e)$ (3). When the algorithm decides that the table contains enough information, it constructs a DFA.

Later, others have made extensions for the L* algorithm to make it work on Mealy machines as well (one example is Maler & Pnueli's algorithm (6)). This allows the algorithm to be used on a larger number of real-world reactive systems (5).

2.3 Tools

Using active model learning algorithms on systems often means you have to build a custom implementation of an algorithm specific to your system. This has a few downsides:

1. The implementation cannot easily be reused for other systems; and
2. Chances are since it is a one-off implementation, it is not optimized to the fullest extent.

Another problem is that many systems have a too large or even infinite number of actions (input options). An example might be a login system where a user can fill in a username and password; the combinations of usernames and passwords are endless and that results in many/infinite states. In such cases, a mapper needs to be placed in between the learner and the teacher that transforms the large set of actions into a small set of abstract actions (7). By using a mapper, a learning algorithm that targets finite-state machines can be applied to these larger systems. However, building a mapper is a cumbersome process, making it even harder to start using automata learning.

Luckily, a few libraries and tools aim to solve these issues by automating the creation of the mapper and optimizing the algorithms as much as possible so that one does not

have to build a custom implementation for every system they work with. Two of these tools stand out: LearnLib and Tomte. LearnLib stands out because it is the most used and mature library for automata learning and implements many different algorithms for learning DFAs and Mealy Machines. Tomte uses LearnLib for learning DFAs and Mealy Machines but is also capable of learning Register Automata. Furthermore, the Tomte tool makes it possible to learn systems with infinite states by abstracting said states. This functionality makes Tomte a logical choice to use on real systems.

2.3.1 LearnLib

LearnLib (8) is a library for active automata learning that has been designed with an emphasis on scalability and performance and is probably the most well-known tool in the field. It was developed at the Chair for Programming Systems at the TU Dortmund University, Germany. It implements several learning algorithms, most of which target DFAs and Mealy machines (like L* and some extensions of L*, but also different algorithms like the TTT (9) or Kearns & Vazirani (10) algorithms). On top of those algorithms, LearnLib also includes an algorithm for non-deterministic finite-state machines (NFAs), namely the NL* algorithm (11). It is similar to the L* algorithm, but some modifications were made to make it work for NFAs. The library has been in development since 2003 and is still being improved upon. The idea is that with LearnLib you can easily start using active automata learning algorithms and equivalence queries and the library provides users with an infrastructure that includes statistics and a SUT adapter.

2.3.2 Tomte

The Tomte tool (4) was created by a team at the Radboud University Nijmegen, The Netherlands. For learning, it uses the LearnLib library, but for some use-cases, it has its own implementation, e.g. for Register Automata. Both tools support DFAs and Mealy machines, but Tomte is also capable of learning Register Automata. A benefit Tomte has over LearnLib is that it automatically generates a mapper, making it more suitable for large systems. Furthermore, most model learning tools can only learn DFAs and Mealy Machines, but most real-world systems are more complex than a simple DFA or Mealy Machine. Therefore, Tomte seems like the tool one should choose to start with model learning on real-world systems as it can abstract systems with infinite states to a form that can be learned by currently existing active learning algorithms.

3

Tomte tool

In this section I shall explain more about how the Tomte tool works, what it was like using the tool, and what could be improved.

3.1 Inner workings

The Tomte tool is placed between the learner and teacher. To make sure the learner can apply an algorithm to the system, three steps are required. First, all the non-determinism of the SUT that is generated by its outputs must be removed. Then a so-called lookahead oracle remembers events with information about the data that have an impact on the future behaviour of the SUT. As a last step, the large set of values of the SUT is abstracted to a small set of symbolic values that the learner is capable of handling. See Figure 3.1 for a diagram of the architecture of Tomte. Sections 3.1.1 - 3.1.3 describe these 3 steps in more detail, but to obtain a complete understanding of the components, one can refer to Aarts, et al. (12).

3.1.1 Determinizer

The determinizer component is placed closest to the teacher and its function is to remove any non-determinism from the SUT for the learner. This is necessary when working with fresh output values, e.g. in a system like a server that generates passwords. The component renames the generated fresh output values to -1, the next one to -2, etc. This is great,

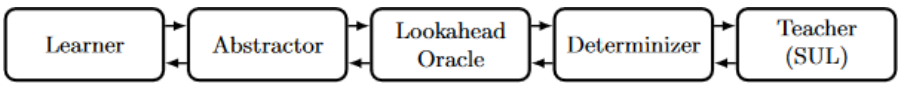

Figure 3.1: Architecture of Tomte

because the output values need to be *neat* and quite often this is not the case, since the learner usually has no control over what the system returns as output. If each output value in a trace is either equal to a previous value or equal to the smallest preceding value minus 1, the trace is considered to be neat. The determinizer is not recommended for systems where collisions can easily occur. A collision is when the automata 'accidentally' selects an output which equals a previous output.

3.1.2 Lookahead oracle

The lookahead oracle is placed right in the middle. It works as follows: the abstractor sends an input action to the oracle and the oracle sends it to the determinizer without altering it. Next, when the oracle receives an output value from the determinizer, it creates a pair consisting of the output value and a valuation. All traces are saved to an observation tree where the root of the tree is the start of the traces. When a new node is added to the tree, the oracle computes a set of variables for it. This set of variables may grow dynamically during the learning process.

3.1.3 Abstractor

During learning, the lookahead oracle sends which variables should be remembered to the abstractor. Based on these variables it will try to create an abstract version for the learner. It will always try to use the smallest possible abstraction. During verification of the hypothesized model the list of variables to remember may be updated if a counterexample is found. If it is updated, Tomte will try again to find a fitting model but with an abstraction created from the new list of variables. We call this method 'counterexample guided abstraction refinement'.

3.1.4 Equivalence checking

Often we cannot perform equivalence queries directly on the SUT. In such cases, a model-based testing tool (MBT tool) should be used (4). An MBT tool will generate a certain amount of test input traces and use these traces on the learner model and the teacher model/SUT. If the output is always equal, the hypothesized model is likely correct or close to what it should be. Otherwise, it will return the output trace for which the equivalence check failed. Suppose some knowledge about the SUT is already available, like an example model in the documentation. In that case, the hypothesized learner model can be compared to the SUT using the CADP toolset and the SUT tool (also created by the team from

Tomte and downloadable from the same website). The SUT tool can simulate a SUT if no real SUT is available. CADP offers a comprehensive set of functionalities covering the entire design cycle of asynchronous systems: specification, interactive simulation, rapid prototyping, verification, testing, and performance evaluation. For verification, it supports the three essential approaches existing in the field: model checking, equivalence checking, and visual checking (13). In Tomte, CADPs equivalence checking features can be used on simulated SUTs to confirm that a learned model is equal to a teacher model.

3.1.5 Additional software

Just like for equivalence checking, some other features of the tool also rely on additional software. It is required to have both the Java SDK and Python 2.7 installed on your machine in order to run Tomte. Furthermore, suppose you want to test Tomte with a simple model before using it on a real system. In that case, you need the UPPAAL (14) tool to create a model visually, after which you can download the model as an XML file which you can use to generate a SUT in Tomte. UPPAAL is a tool to design, simulate and verify models in a visual interface. On top of that, Graphviz (15) is required to generate models in the .dot format (.dot is a graph description language).

3.2 Commands

The Tomte tool is a command-line interface (CLI). It includes a couple of useful commands that automate different actions one might need to perform when using automata learning.

3.2.1 sut_run

To start using Tomte, you need to have a running SUT and the 'sut_run' command takes care of that. You can use any SUT that is capable of communication over TCP/IP network sockets. The positional 'modelfile' argument is required and it describes the path to your input UPPAAL model XML file that you might use while testing Tomte or the path to your SUT in a real-world system. On top of that, a couple of optional parameters are available as well:

- -h/–help: for help about the usage/available arguments;
- -v/–verbose: to print all inputs and outputs; and
- –port PORT: to define a specific TCP port to listen on for incoming connections.

3.2.2 tomte_learn

When the SUT is running, the 'tomte_learn' command can be used to learn a model by running the actual Tomte tool. It requires a config.yaml file and a sutinfo.yaml file (see next section). The arguments available to this command include one positional argument (required) and seven optional arguments. The positional argument is the 'configfile' argument, which describes a path to a .yaml file which contains learner configuration options. The optional arguments are the 'help', and 'port' arguments which function as described in the previous section, and the other five optional arguments are the following:

- –seed SEED: seed to use for random number generation (different seeds can produce different results, but by using a seed number you can easily rerun a previously run learning process that will produce the same results every time).
- –max-memory MAX_MEMORY: maximum memory to use by virtual machine in GB. There is not much documentation on this argument, but it is probably useful on less powerful systems so they do not crash.
- –output-dir OUTPUT_DIR: the tool generates a lot of new files and they need to be saved somewhere. By default it is in a folder called 'output', but you can override that with a path of your own choosing by using this parameter.
- -e/–eclipse: run the tool using build code from the eclipse project. Again, this command does not include much information, so the benefits of running the command with this argument are unclear.
- –config-option CONFIG_OPTION: overrule a configuration option from the config file.

3.2.3 sut_uppaal2sutinfo

A sutinfo yaml file can be generated from the teacher model with the 'sut_uppaal2sutinfo' command. The sutinfo file describes some information in a yaml config file, like what the input and output interfaces of the SUT are and the SUT name (the name is solely used for the output folder name and is not required for learning). The 'sut_uppaal2sutinfo' command has only one optional parameter, namely the '-h' or '–help' parameter which simply outputs some useful information about using this command. It has two positional arguments; the first one is used for selecting the teacher model file and the second one is the name of the sutinfo file. This command is useful when testing how Tomte works, but

in a real-world system you would need to configure this file yourself as you do not have an exact model of the system yet. Luckily, it is an extremely easy to understand file, so creating it should not be too hard.

3.2.4 sut_uppaal2layoutformat

If you want to generate a pdf from the learned concrete UPPAAL model, you can use this command. It is required to also have the Python module 'pygraphviz' installed.

3.2.5 sut_compare

Finally, the 'sut_compare' command can confirm that the learned model is equal to the teacher model. It first flattens both models to CADP .bcg files and then runs the CADP bisimulator, which compares the two models using a specified equivalence relation. The bisimulator reformulates the graph comparison problem as a boolean equation system. It can also generate a counterexample, which is helpful to see why two models are not equal [1]. The command has four possible positional arguments: model1, model2, range_min, range_max. 'model1' and 'model2' are the two models that will be compared and are therefore required. The models can be an UPPAAL model, CADP .aut, JTorX .aut or .dot format. 'range_min' and 'range_max' describe the minimum and maximum value of range used for input parameters when flattening (see Section 3.2.6) the model. These are only needed for models with data and both arguments already have a default value of 0. Next to the optional arguments -h, –help, -v, –verbose which were already described in the sut_run section, there exist also some special optional arguments:

- –keep-tmp-files: useful if you want to keep the temporary files that were generated for doing the comparison.
- –skip-both-ways: for IOMEALY models, the two models are compared both ways to find different outputs for both models. You can decide to only compare model1 against model2 with this argument.
- –output-json, –output-boolean, –output-text: by default, the command outputs in json style the result of the comparison with some additional data. If you only want to see the result, –output-boolean should be used. If you dislike the json format and prefer a text format, –output-text should be used.

[1] `https://cadp.inria.fr/tools.html#section-4`

- –lts, –io-mealy: for LTS automata it is required to specify it is a label transition system. By default, the command will assume it is an I/O automaton describing a Mealy automaton.

3.2.6 Flattening

We can remove data parameters from input actions by defining a set of symbols for each input where each symbol represents a unique instance of its input parameters from the given range. This function is part of the SUT tool and requires the CADP toolset. It includes three conversion commands: sut_uppaal2bcg, bcg_io and sut_ioaut2mealydot. 'sut_uppaal2bcg' flattens a UPPAAL model to a bcg (binary coded graph) file [1]. Then the bcg_io command can be used to transform the bcg file to an .aut (aldebaran) format [2]. Finally, with the 'sut_ioaut2mealydot' command one can transform the .aut file to a .dot file. Combining these commands will transform the UPPAAL model to a Mealy model.

3.3 Personal experience

Tomte is simple to use once you know how it works, but there are definitely elements that could make it easier to get started with the tool. The program relies on five different other programs, of which four are hard requirements. Two of these five programs require a licence (free academic licence or paid commercial licence with no pricing available on their sites). After downloading all required software, the Tomte tool and optionally the SUT tool, you can start the installation. Tomte should (according to their website) work on Linux, Mac and Windows. However, when I first tried to install Tomte on my Windows machine, the sut_run command did accept any arguments, rendering the tool useless. Installing it on Linux did work after manually adding Tomte and the SUT tool to the PATH variable (this is required to specify where Tomte and the SUT tool are located, so the different commands run correctly). If you would like to use the CADP toolset for equivalence checking, you have to add CADP to the PATH variable and create a CADP_LANGUAGE (value: english) variable to make it work with Tomte, and you have to email the maintainers to retrieve a licence key. Depending on where you installed the software, these variables should be added to either the ' /.bashrc' or '/etc/profile' file. To put it concisely, the installation process is quite painful.

[1] `https://cadp.inria.fr/man/bcg.html`

[2] `https://www.mcrl2.org/web/user_manual/language_reference/lts.html#aldebaran-format`

Using the tool is easy, but the documentation is lacking. It gives enough information to run the example models that come with downloading Tomte and a little information is given on how to generate a SUT of your own which you can use for testing Tomte, but there is no information available about how to use Tomte for a real-world system. Without diving in all of the files and reading all the code of the example models, it is unclear how one could attach Tomte to an actual system. The error messages are not descriptive and the stack traces do not help much either, since you do not have access to the source code of Tomte. This makes debugging your SUT quite hard.

4

Experiments

Tomte comes with 34 example models, so new users can look at how Tomte works before using it on their own systems. All of these models have their own SUT that is generated from an UPPAAL file. In this section I will explore how these SUTs are created and what kind of output is generated from the learning process. To make sure we get results that we can analyze, the 'Making your own SUT' guide on the Tomte website [1] is followed closely. Therefore, the SUT we will create in this section will be for a Register Automaton of an alternating bit protocol receiver [2] to observe what type of output is generated. We will also attempt to build a custom SUT by modifying the fresh_cav_multilogin2 example model so it functions as an external API, to make it a more realistic system.

4.1 Generating a SUT

The first step necessary for creating a SUT is to draw the teacher model in UPPAAL. In UPPAAL, you can easily add different states and transitions with a drag-and-drop interface. Then, by clicking a transition, you can open a popup which gives you the option to add guards, the name of the accompanying method and update functions. It is important to note that you have to follow a certain syntax. Input methods should start with an uppercase "I" and output methods should start with an uppercase "O". Figure 4.1 displays the UPPAAL interface.

We also have to write some declarations. Tomte requires these declarations to be written in the global declarations section and thus not in the local declarations section. This has probably something to do with how Tomte parses the UPPAAL XML file. If the local

[1] `https://tomte.cs.ru.nl/Sut-0-4/MakingSut`

[2] `https://www.d.umn.edu/~gshute/net/reliable-data-transfer.xhtml`

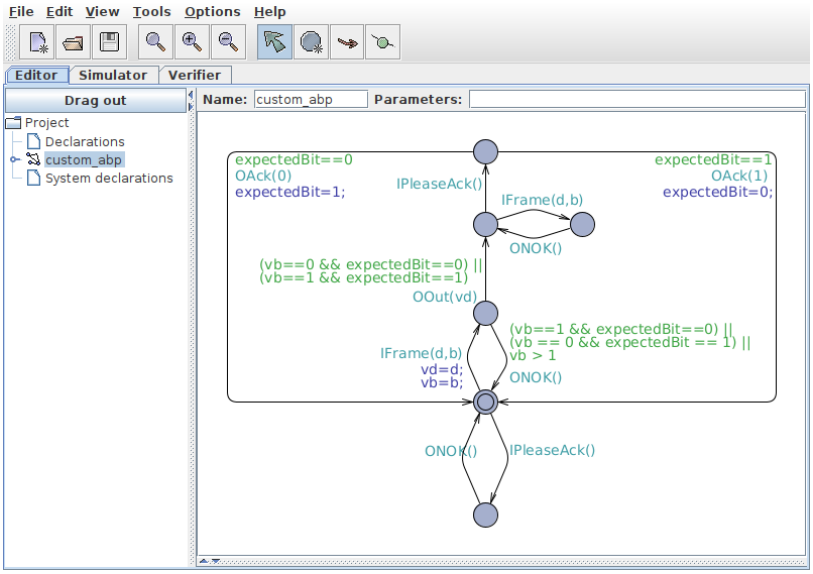

Figure 4.1: Creating a model in UPPAAL

section is used instead of the global section, the command sut_uppaal2sutinfo will not work. Even if you write the sutinfo.yaml file manually, this could probably cause some issues with the sut_run command as well. An overview of the declaration can be seen in Figure 4.2. When we press the save button, an XML file with all the settings is generated and saved to your computer.

Now that we have the teacher model, we need to create a sutinfo.yaml file, so the SUT tool can generate a SUT implementation. We can do this by running the sut_uppaal2sutinfo command. The sutinfo.yaml file is basically a yaml representation of the UPPAAL declarations (see Appendix A). If the declarations are incomplete or contain errors, this command will detect it, which is useful for debugging. Everything is now ready for the SUT to work.

4.2 Learning

In order to start learning, we need to create a config.yaml file. This file contains options for the learning/testing process. While some of the options are easily understood, other options would require documentation. However, there is nearly no documentation on this file. The only information about the configuration options on the Tomte website tells us that the learning section of the file specifies the parameters that are needed for generating a hypothesis during learning and that the testing section specifies the parameters that are needed for testing a hypothesis against the SUT. While there are some options that are

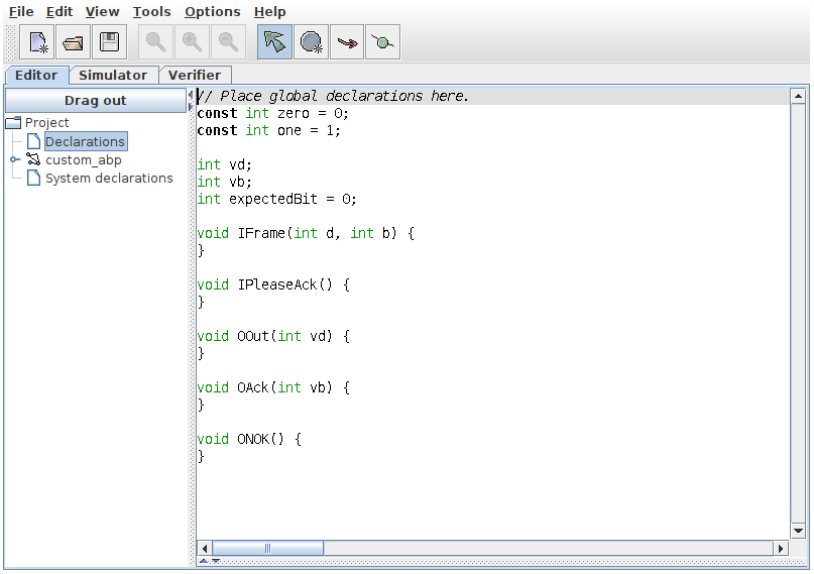

Figure 4.2: Adding declarations in UPPAAL

understandable, the lack of documentation makes it hard to understand what every option does. Luckily, the example models all include this file as well and they do not differ much, so you can copy one of those and if needed tweak some options, like the seed or perhaps some of the options in the 'testing' block. The configuration that was used for obtaining the results below can be found in Appendix B.

We should now start up the SUT with the sut_run command. At this point, there was a typo in one of the guards in the UPPAAL model ('expectedBitt' instead of 'expectedBit'). The sut_run command detected that a variable was used in a guard that does not exist and returned an error describing the issue in enough detail. After the typo was fixed, sut_run ran without any more issues. To begin learning the model, we need to open a second terminal window and run the tomte_learn command. This command will output everything it does in the terminal and when finished, it will create an output folder in which statistics about the various runs (a run is everything that happens from the moment Tomte starts sending queries until Tomte is finished with the equivalence queries, and if a counterexample is found, a new run is started) and the learned model files are stored. Finally, a check is done with the sut_compare command to check whether the learned model is correct by comparing it to the teacher model.

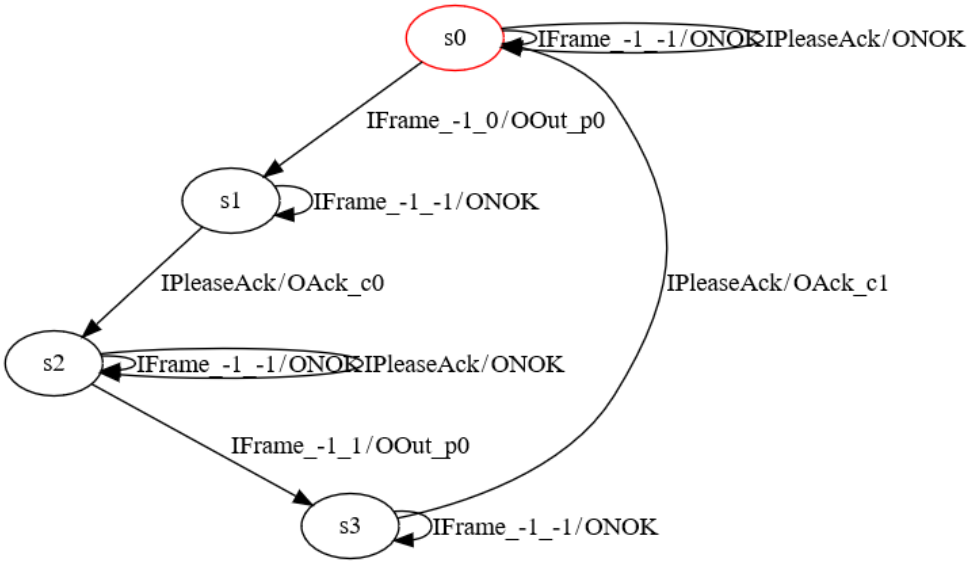

Figure 4.3: Learned abstract model

4.3 Results

According to the result of the sut_compare command, our learned model is equal to the teacher model (Appendix E). The output folder also contains interesting data, like exhaustive statistics about the runs of the program (Appendix C), the learned model in UPPAAL format (but without positioning, thus making it hard to see the states and transitions when viewing the model in UPPAAL without first editing the file), the learned abstraction that was generated for the learning algorithm (Appendix F) with a accompanying .dot file (see Figure 4.3 for a visualisation of the abstraction), and a folder with for each run the hypothesis that was made. In the statistics file, it is logged how many runs it took Tomte to find the correct model, how many queries were made, which counterexamples were found, and moreover for each run and at the end of the file is a summary of all the data combined.

4.4 Using Tomte for real-world systems

In the real world, this method for generating a SUT would not work, since you do not have the exact model yet. While there is not really any documentation on how to connect Tomte to a real system (even though one of the goals of Tomte is to bridge the gap between automata learning and real-world systems), it is possible to gain some knowledge by looking at the fresh_cav_multilogin2 example model. This model uses a small Java application as SUT and the main logic of the system it simulates is written in the Sut.java file. Thus,

by modifying this file, we should be able to connect Tomte to a real system. I built an implementation that connects a custom SUT with an external API [1].

To do this, I wrote a simple API in Python (Appendix I) that has the same functionality as the example model. It has four input methods: IRegister, ILogin, ILogout and IChangePassword. IRegister has one parameter called uid, which checks whether a user already exists for that uid and if not, creates a user and generates a password which is returned to the API caller. This password can then be used by the user to call ILogin (also with the uid, of course) and a user can only log in if they are currently logged out. If a user is logged in, they can call ILogout and IChangePassword with the uid. I also had the create an extra 'reset' method, to remove all data to return to the default state of the program.

Now that the API is set up and running, we should modify the Sut.java file (Appendix G). Currently, this file includes all logic of the API, so we start by removing all that logic. Instead, we create a method for each available API call, which takes as input the uid (and in case of the ILogin method, also the password). Then, in the method we send a request to the API. In case of the IRegister and IChangePassword calls, a password is returned, so we return the password as parameter to our learning algorithm together with the output method name 'OOK'. If an API call is unsuccessful, we return the method name "ONOK" with no parameters. We also create a reset method which just calls the reset endpoint of the API without returning any data. A handler() method redirects each method of the learning algorithm to the correct method in the SUT. ILogin and ILogout should also return a parameter with OOK, but this value is not important for the program. Therefore, we return a random integer as fresh value. Next, we generate a .jar file and the SUT is finished. Now, our learning algorithm can fully contact the API.

We start learning with 'tomte_learn'. What is interesting is, even though the behaviour of the program should be identical to the behaviour of the original example model, some errors are returned. The first one that had to be dealt with, was an error regarding that the program ran too long. This could be fixed by adding a 'max_time' value to our config.yaml (Appendix H) file, with a high value, e.g. 3000 (in seconds). The next error is probably caused by the fresh values returned by ILogin and ILogout:

```
abslearning.exceptions.ValueCannotBeDecanonizedException:
value -1100 cannot be decanonized
```

[1] https://github.com/GideonRoose/model-learning-experiment

As was mentioned in Section 3.1.1, the determinizer deals with fresh values by transforming them into a canonical form. When these values later have to be decanonicalized, it can cause errors. Perhaps Tomte is unable to correctly differentiate between fresh values that should be saved and reused and values that should be thrown away. Interestingly, the value that could not be decanonicalized was always lower than or equal to -1000. So maybe specific values are canonicalized to canonical values lower than or equal to -1000, meaning that the error would only be present for these specific values. The real issue, however, is unclear, so we can only speculate about what causes the error.

In the end, it is possible to create a functional SUT, but it is next to impossible to debug its errors. For example, the above error could be caused by something in my SUT implementation, but it could also be caused by how Tomte deals with these fresh values. Since you can not read the source code of Tomte and thus the implementation of the determinizer, it is impossible to understand how the determinizer is implemented fully.

5

Conclusion

Model learning can be beneficial for discovering whether a program conforms to the appropriate standards and determining how a system functions. It should also be possible to use it on real-world systems instead of just theoretical ones, but the available tools do not provide the user with enough information to get this done. As I experienced in my own implementation, connecting a tool like Tomte to even a simple external API is difficult. I had to thoroughly understand the tool and its inner workings to use it, and even then, my knowledge appeared insufficient to obtain valuable results. Modifying the example SUT to fit my needs was a pain as it was initially unclear which files should be modified. Only after carefully examining the code in all the different files was it possible to build a custom SUT.

To make Tomte applicable to real-world systems, a few items should be addressed. Firstly, the documentation must be extended. It is currently at a state where it is possible to make theoretical models and see how the tool functions, but more information should be provided to users on how to implement their own SUT. Now, the only method they have is to copy the custom SUT from one of the example models and modify that one to fit your needs. Modifying the example model requires you to know Java, and you have to reread the example code multiple times to get an idea of what is happening. Then still, you do not have enough information to debug your SUT in case of problems. A better approach would be to define a structure in which your SUT should function. Which endpoints should be available, how the socket server should be set up, and what data should be returned to Tomte. Defining such a structure would make it possible for users to more easily implement a SUT in a language of their choosing and thus make it easier for users to know what is going wrong with errors.

Furthermore, error messages returned by Tomte in the learning process should be documented better. Right now, it is probably enough for the developers of Tomte to know what is going wrong. However, the ordinary user cannot find all the necessary information to debug their SUT. A great addition would be proper documentation on the determinizer. The goal of the determinizer is described in one paper. However, since errors related to the determinizer can quickly occur, as I have seen in my implementation, it could help users gain a deeper understanding of how Tomte deals with fresh values by describing the determinizer's functionality in detail. Finally, the config.yaml file should be explained in more detail, so users know what options they can edit to fit their needs.

In conclusion, the tool could be powerful for actively learning automata if the previous points are addressed. It is not mature enough to be easily implemented in real systems, but it has the potential. Model learning is a valuable method for learning automata, and it would be interesting to see others adopt tools such as Tomte when possible.

References

[1] Frits Vaandrager. **Model learning**. *Communications of the ACM*, **60**(2):86–95, 2017. 1

[2] Maik Merten. *Active automata learning for real life applications.* PhD thesis, Technical University of Dortmund, 01 2013. 2

[3] Dana Angluin. **Learning regular sets from queries and counterexamples**. *Information and Computation*, **75**(2):87–106, 1987. 2, 5, 6

[4] Fides Dorothea Aarts. *Tomte: bridging the gap between active learning and real-world systems.* PhD thesis, [Sl: sn], 2014. 2, 7, 10

[5] Bernhard Steffen, Falk Howar, and Maik Merten. **Introduction to active automata learning from a practical perspective**. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 256–296. Springer, 2011. 3, 6

[6] Oded Maler and Amir Pnueli. **On the learnability of infinitary regular sets**. *Information and Computation*, **118**(2):316–326, 1995. 6

[7] Fides Aarts, Falk Howar, Harco Kuppens, and Frits Vaandrager. **Algorithms for inferring register automata**. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 202–219. Springer, 2014. 6

[8] Malte Isberner, Falk Howar, and Bernhard Steffen. **The Open-Source LearnLib**. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, pages 487–495, Cham, 2015. Springer International Publishing. 7

[9] Malte Isberner, Falk Howar, and Bernhard Steffen. **The TTT algorithm: a redundancy-free approach to active automata learning**. In *International Conference on Runtime Verification*, pages 307–322. Springer, 2014. 7

[10] Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994. 7

[11] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. **Angluin-Style Learning of NFA.** In *IJCAI*, **9**, pages 1004–1009, 2009. 7

[12] Fides Aarts, Paul Fiterau-Brostean, Harco Kuppens, and Frits Vaandrager. **Learning register automata with fresh value generation**. In *International Colloquium on Theoretical Aspects of Computing*, pages 165–183. Springer, 2015. 9

[13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. **CADP 2011: a toolbox for the construction and analysis of distributed processes**. *International Journal on Software Tools for Technology Transfer*, **15**(2):89–107, 2013. 11

[14] Alexandre David, Kim Guldstrand Larsen, Gerd Behrmann, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. **UPPAAL 4.0**. In *Third International Conference on the Quantitative Evaluation of SysTems (QEST) 2006*, pages 125–126. IEEE Computer Society Press, 2006. 11

[15] Emden R. Gansner and Stephen C. North. **An open graph visualization system and its applications to software engineering**. *SOFTWARE - PRACTICE AND EXPERIENCE*, **30**(11):1203–1233, 2000. 11

Appendix

A Contents of sutinfo.yaml (input)

```
constants:
- 0
- 1
inputInterfaces:
  IFrame:
  - int
  - int
  IPleaseAck: []
name: custom_abp
outputInterfaces:
  OAck:
  - int
  ONOK: []
  OOut:
  - int
```

B Contents of config.yaml (input)

B.1 Options used for overwriting defaults

```
learning:
    mode: "stateVarN"
    sutinfoFile: "sutinfo.yaml"
    seed: 12345
    preferFirst: false
testing:
    minValue: 0
```

```
    maxValue: 400
    maxTraceLength: 150
    maxNumTraces: 500
verification:
    method: null
```

B.2 All options used in tomte_learn

```
currentOutputDir: output/22-06-25_09.54.06_custom_abp/
learning:
    mode: stateVarN
    relations: [ 'EQUAL', ]
    sutinfoFile: /home/user/tomte-0.41/models/custom_abp/sutinfo.yaml
    seed: 12345
    generateHypCode: false
    reuseCounterExample: true
    reduceCounterExample: true
    algorithm: observationPack
    freshValueStep: 100
    memVOrder: lookahead
    reconstructAbstractions: false
    reductionStrategies: [ 'loop', 'singleTransition', ]
    useSutSimulation: false
    maxTime: 0
    maxNumRuns: -1
testing:
    maxNumTraces: 500
    minValue: 0
    maxValue: 400
    valuesExtendable: true
    useTestingFallback: true
    useConstantsInTestInputs: true
    drawFresh: 0.1
    methods: [ 'randomWalkFromState', ]
    minTraceLength: 5
    avgTraceLength: 10
    maxTraceLength: 150
    randomLength: 10
    testTraces: [ ]
```

```
verification:
    when: atEnd
    cadp:
learning:
    compareMinValue: 0
    compareMaxValue: 2
    dataStructureElements: 0
sutInterface:
    communicationChannel: socket
    socket:
        portNumber: 9999
        server: unknown
    directCall:
        package: generated.sut
        className: SutImpl
sutImplementation:
    modelName: ${modelDirName}
    projectClassPath: build
sutSimulation:
    method: DirectMethodCall
learnResults:
    outputDir: output/22-06-25_09.54.06_custom_abp
    abstractModelDotFile: learnedAbstractModel.dot
    abstractModelPdfFile: learnedAbstractModel.pdf
    writeAbstractModelPdfFile: false
    learnedConcreteModelFile: learnedConcreteModel.xml
    abstractionFile: learnedAbstraction.txt
    learnedModelDataFile: learnedModelData.json
    statisticsFile: statistics.txt
    statisticsJsonFile: statistics.json
logging:
    logDir: log/
    hypDir: generated/hypothesis/
    logFile: log.txt
    logFileThreshold: off
    consoleThreshold: info
    rootLoggerLevel: info
    learnlibLogLevel: info
```

```
    special:
        hypotheses: true
        hypothesesWritePdf: false
        memQueries: false
        memTraces: false
        equivQueries: false
        equivTraces: false
        memQueriesFile: memQueries.txt
        memTracesFile: memTraces.txt
        equivQueriesFile: equivQueries.txt
        equivTracesFile: equivTraces.txt
        concreteTree: false
        concreteTreeFile: concreteTree.dot
        concreteTreeStatistics: false
        concreteTreeStatisticsFile: concreteTreeStatistics.json
        plotConcreteTreeStatistics: false
params:
    originalArgs: [ '--tomte-root-path', '/home/user/tomte-0.41',
                    '--output-dir', 'output/22-06-25_09.54.06_custom_abp',
                    '--port', '9999',
                    '/home/user/tomte-0.41/models/custom_abp/config.yaml', ]
    configFile: /home/user/tomte-0.41/models/custom_abp/config.yaml
    tomteRootPath: /home/user/tomte-0.41
    verboseLevel: 0
dependency:
    pythonCmd: /usr/bin/python
    javaCmd: /usr/bin/java
    javacCmd: javac
    tomteLearnresult2uppaalCmd: /home/user/tomte-0.41/bin/tomte_learnresult2uppaal
    sutUppaal2JarCmd: /home/user/sut-0.41/bin/sut_uppaal2jar
    sutUppaal2SutInfoCmd: sut_uppaal2sutinfo
    sutRunCmd: sut_run
devel:
    printStackTraceOfExceptions: true
    runInEclipse: false
```

C Contents of statistics.txt (output)

```
--------------------------------------------------------------------------------
                          RUNS
--------------------------------------------------------------------------------
Run number                                  : 1
Start learning from scratch                 : true
HypL index/#states after run                : 1/1
Membership queries                          : 9
Membership inputs                           : 12
Running time of membership                  : 82 ms = 0:00:00 (h:m:s)
Testing equivalence queries                 : 1
Testing equivalence inputs                  : 6
Running time of testing                     : 40 ms = 0:00:00 (h:m:s)
Ce analysis membership queries              : 4
Ce analysis membership inputs               : 24
Running time of counterexample analysis     : 108 ms = 0:00:00 (h:m:s)

Learn resets                                : 9
Learn inputs                                : 10
Testing resets                              : 2
Testing inputs                              : 12
Ce Analysis resets                          : 8
Ce Analysis inputs                          : 42

Number nodes in observation tree            : 7
Number end nodes in observation tree        : 4

num Ce inputs EquivOracle                   : 12
num Ce inputs AfterLoopReduction            : 12
num Ce inputs AfterCeAnalysis               : 0
number Of Reused CounterExamples            : 0

Abstraction refinement done                 : 0
Store updates on CE                         : 0
Store updates on inconsistency              : 0
Counterexample sent to learner              : 0
Description of run                          :
```

```
 Termination Reason: concrete counterexample found!

    concrete CE inputs:
           [IPleaseAck(), IFrame(100,200), IPleaseAck(), IPleaseAck(),
           IFrame(200,200), IFrame(0,0)]
    executed:
       on Sut (with abstractions determined using statevars of sut):
           IPleaseAck() / ONOK() -> IFrame_-1_-1(100,200) / ONOK() ->
           IPleaseAck() / ONOK() -> IPleaseAck() / ONOK() ->
           IFrame_-1_-1(200,200) / ONOK() -> IFrame_-1_-1(0,0) / OOut_c0(0)
       on Hyp (with abstractions determined using statevars of hyp):
           IPleaseAck() / ONOK() -> IFrame_-1_-1(100,200) / ONOK() ->
           IPleaseAck() / ONOK() -> IPleaseAck() / ONOK() ->
           IFrame_-1_-1(200,200) / ONOK() -> IFrame_-1_-1(0,0) / ONOK()
trace on sut

IPleaseAck() /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_-1(100,200) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IPleaseAck() /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IPleaseAck() /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_-1(300,400) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_-1(500,0) /
OOut_p0(500)
   memV: []   stateVarUpdates: {}   stateVars: {}



trace on hyp
```

```
IPleaseAck() /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(100,200) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IPleaseAck() /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IPleaseAck() /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(300,400) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(500,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
```

```
--------------------------------------------------------------------------------
Run number                                    : 2
Start learning from scratch                   : true
HypL index/#states after run                  : 2/3
Membership queries                            : 40
Membership inputs                             : 102
Running time of membership                    : 94 ms = 0:00:00 (h:m:s)
Testing equivalence queries                   : 2
Testing equivalence inputs                    : 22
Running time of testing                       : 18 ms = 0:00:00 (h:m:s)
Ce analysis membership queries                : 4
Ce analysis membership inputs                 : 40
Running time of counterexample analysis       : 60 ms = 0:00:00 (h:m:s)

Learn resets                                  : 40
```

APPENDIX.

```
Learn inputs                                : 74
Testing resets                              : 4
Testing inputs                              : 38
Ce Analysis resets                          : 11
Ce Analysis inputs                          : 100

Number nodes in observation tree            : 38
Number end nodes in observation tree        : 23

num Ce inputs EquivOracle                   : 20
num Ce inputs AfterLoopReduction            : 20
num Ce inputs AfterCeAnalysis               : 0
number Of Reused CounterExamples            : 0

Abstraction refinement done                 : 0
Store updates on CE                         : 0
Store updates on inconsistency              : 0
Counterexample sent to learner              : 0
Description of run                          :

 Termination Reason: concrete counterexample found!

    concrete CE inputs:
           [IFrame(100,0), IFrame(200,0), IFrame(0,0), IFrame(1,0),
           IFrame(200,300), IFrame(1,0), IFrame(200,0), IPleaseAck(),
           IPleaseAck(), IFrame(0,1)]
    executed:
       on Sut (with abstractions determined using statevars of sut):
           IFrame_-1_0(100,0) / OOut_p0(100) -> IFrame_-1_0(200,0) / ONOK() ->
           IFrame_-1_0(0,0) / ONOK() -> IFrame_-1_0(1,0) / ONOK() ->
           IFrame_-1_-1(200,300) / ONOK() -> IFrame_-1_0(1,0) / ONOK() ->
           IFrame_-1_0(200,0) / ONOK() -> IPleaseAck() / OAck_c0(0) ->
           IPleaseAck() / ONOK() -> IFrame_-1_-1(0,1) / OOut_c0(0)
       on Hyp (with abstractions determined using statevars of hyp):
           IFrame_-1_0(100,0) / OOut_p0(100) -> IFrame_-1_-1(200,0) / ONOK() ->
           IFrame_-1_-1(0,0) / ONOK() -> IFrame_-1_-1(1,0) / ONOK() ->
           IFrame_-1_-1(200,300) / ONOK() -> IFrame_-1_-1(1,0) / ONOK() ->
           IFrame_-1_-1(200,0) / ONOK() -> IPleaseAck() / OAck_c0(0) ->
```

```
          IPleaseAck() / ONOK() -> IFrame_-1_-1(0,1) / ONOK()
trace on sut

IFrame_-1_0(100,0) /
OOut_p0(100)
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_0(200,0) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_0(400,0) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_0(500,0) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_-1(600,300) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_0(700,0) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_0(800,0) /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IPleaseAck() /
OAck_c0(0)
   memV: []   stateVarUpdates: {}   stateVars: {}
IPleaseAck() /
ONOK()
   memV: []   stateVarUpdates: {}   stateVars: {}
IFrame_-1_-1(900,1) /
OOut_p0(900)
   memV: []   stateVarUpdates: {}   stateVars: {}



trace on hyp
```

```
IFrame_-1_0(100,0) /
OOut_p0(100)
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(200,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(400,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(500,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(600,300) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(700,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(800,0) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IPleaseAck() /
OAck_c0(0)
   memV: null   stateVarUpdates: null   stateVars: {}
IPleaseAck() /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
IFrame_-1_-1(900,1) /
ONOK()
   memV: null   stateVarUpdates: null   stateVars: {}
```

```
Run number                                  : 3
Start learning from scratch                 : true
HypL index/#states after run                : 3/4
```

```
Membership queries                              : 85
Membership inputs                               : 268
Running time of membership                      : 145 ms = 0:00:00 (h:m:s)
Testing equivalence queries                     : 500
Testing equivalence inputs                      : 5799
Running time of testing                         : 1136 ms = 0:00:01 (h:m:s)
Ce analysis membership queries                  : 0
Ce analysis membership inputs                   : 0
Running time of counterexample analysis         : 0 ms = 0:00:00 (h:m:s)

Learn resets                                    : 85
Learn inputs                                    : 144
Testing resets                                  : 501
Testing inputs                                  : 5809
Ce Analysis resets                              : 0
Ce Analysis inputs                              : 0

Number nodes in observation tree                : 99
Number end nodes in observation tree            : 56

num Ce inputs EquivOracle                       : 0
num Ce inputs AfterLoopReduction                : 0
num Ce inputs AfterCeAnalysis                   : 0
number Of Reused CounterExamples                : 0

Abstraction refinement done                     : 0
Store updates on CE                             : 0
Store updates on inconsistency                  : 0
Counterexample sent to learner                  : 0
Description of run                              : no counterexample found

-------------------------------------------------------------------------------
                            SUMMARY
-------------------------------------------------------------------------------
Total running time                              : 547 ms = 0:00:01 (h:m:s)
               '-> with last run                : 1683 ms = 0:00:02 (h:m:s)
Total running time of Membership                : 321 ms = 0:00:00 (h:m:s)
Total running time of Testing                   : 58 ms = 0:00:00 (h:m:s)
```

```
                        '-> with last run    : 1194 ms = 0:00:01 (h:m:s)
Total running time of CounterExample analysis  : 168 ms = 0:00:00 (h:m:s)

Total processLearningResultTime                      : 17 ms = 0:00:00 (h:m:s)
Total verificationTime                               : 0 ms = 0:00:00 (h:m:s)

Total realTime                                 : 2222 ms = 0:00:02 (h:m:s)
real start time                                : 1656143646522 ms since Jan 1, 1970 GMT
                                                 = 2022-06-18 09:54:06
real end time                                  : 1656143648744 ms since Jan 1, 1970 GMT
                                                 = 2022-06-18 09:54:08

Total runs (include last test run)             : 3
Total membership queries                       : 134
Total membership inputs                        : 382
Total Testing equivalence queries              : 3
                  '-> with last run            : 503
Total Testing equivalence inputs               : 28
                  '-> with last run            : 5827
Total Ce Analysis membership queries           : 8
Total Ce Analysis membership inputs            : 64
Total abstraction refinements                  : 0
Total number of guard lookahead traces added   : 0
Total number of output lookahead traces added  : 0
Total number of counterexamples sent to learner: 0
Total states in learned abstract Mealy machine : 4

Learn resets                                   : 134
Learn inputs                                   : 228
Testing resets                                 : 6
Testing inputs                                 : 50
Ce Analysis resets                             : 19
Ce Analysis inputs                             : 142

Number nodes in observation tree               : 99
Number end nodes in observation tree           : 56

num Ce traces EquivOracle                      : 2
```

```
num Ce inputs EquivOracle                    : 32
num Ce traces AfterLoopReduction             : 2
num Ce inputs AfterLoopReduction             : 32
num Ce traces CeAnalysis                     : 0
num Ce inputs AfterCeAnalysis                : 0
number Of Reused CounterExamples             : 0

Termination Reason                           : Couldn't find counterexample
                                               after maxNumTraces applied
Learning Successfull                         : null

counter examples:
run 0: IPleaseAck() / ONOK_?() -> IFrame_?_?(88,379) / ONOK_?() ->
       IPleaseAck() / ONOK_?() -> IPleaseAck() / ONOK_?() ->
       IFrame_?_?(379,379) / ONOK_?() -> IFrame_?_?(0,0) / ONOK_?()
run 1: IFrame_?_?(22,0) / OOut_?(22) -> IFrame_?_?(245,0) / ONOK_?() ->
       IFrame_?_?(0,0) / ONOK_?() -> IFrame_?_?(1,0) / ONOK_?() ->
       IFrame_?_?(245,273) / ONOK_?() -> IFrame_?_?(1,0) / ONOK_?() ->
       IFrame_?_?(245,0) / ONOK_?() -> IPleaseAck() / OAck_?(0) ->
       IPleaseAck() / ONOK_?() -> IFrame_?_?(0,1) / ONOK_?()
```

D sut_run

D.1 Without verbose mode

```
$ sut_run model.xml --port 9999

SUT simulation socketserver
-> listening at port : 9999
-> verbose mode : OFF
-> the server has a timeout of 30 seconds
   note: to prevent unnecessary servers to keep on running

New client...
Starting client...
New client...
New client...
Starting client...
New client...
```

```
Starting client...
New client...
Starting client...
Starting client...
New client...
Starting client...
Closing client...
Closing client...
Closing client...
Closing client...
Closing client...
Closing client...
```

D.2 Partial output with verbose mode

```
...
input: IFrame_109_1
output: ONOK
input: IFrame_1_1
output: ONOK
input: IFrame_0_1
output: ONOK
input: IFrame_109_109
output: ONOK
input: IFrame_1_1
output: ONOK
input: IPleaseAck
output: ONOK
input: IFrame_0_0
output: OOut_0
input: IFrame_1_109
output: ONOK
input: IFrame_109_1
output: ONOK
input: IFrame_109_0
output: ONOK
input: reset
reset sut
input: IFrame_373_0
```

```
output: OOut_373
input: IPleaseAck
output: OAck_0
input: IFrame_373_1
output: OOut_373
input: IFrame_0_1
output: ONOK
input: IFrame_0_1
output: ONOK
input: IFrame_373_1
output: ONOK
input: IPleaseAck
output: OAck_1
input: IFrame_390_0
output: OOut_390
input: IPleaseAck
output: OAck_0
input: IFrame_390_1
output: OOut_390
input: IFrame_1_0
output: ONOK
input: IFrame_1_0
output: ONOK
input: IPleaseAck
output: OAck_1
input: reset
reset sut
...
```

E sut_compare

```
$ sut_compare output/22-06-18_09.54.06_custom_abp/learnedConcreteModel.xml \
tomte-0.41/models/custom_abp/model.xml
{
    "equal": true,
    "fullOutput": "flatten first model to a cadp bcg file\n-------------
                  ----------------------------------\n\n\"model.bcg\" =
```

```
                generation of \"model.lotos\"\n    (* 10 states, 14
                transitions, 2.4 Kbytes *)\n\nflatten second model to
                a cadp bcg file\n-----------------------------------
                ----------\n\n\"model.bcg\" = generation of
                \"model.lotos\"\n    (* 10 states, 14 transitions,
                2.4 Kbytes *)compare model1.bcg with model2.bcg with
                cadp caesar\n-----------------------------------------
                ------\nbcg_open: using ‘‘/home/gideon/cadp//bin.x64/
                bisimulator.a’’\nbcg_open: running ‘‘bisimulator -diag
                -bfs -strong model2.bcg’’ for ‘‘./model1.bcg’’\n\nTRUE",
    "inputsCounterExample": null,
    "outputsModel1": null,
    "outputsModel2": null
}
```


F Learned abstraction

```
-----------------------------------------------
Concrete alphabet:
IFrame(int, int)
IPleaseAck()
OAck(int)
ONOK()
OOut(int)
-----------------------------------------------
Abstraction:
IFrame([], [c0, c1])
IPleaseAck()
OAck()
ONOK()
OOut()
-----------------------------------------------
Abstract alphabet:
IFrame_-1_-1
IFrame_-1_0
IFrame_-1_1
IPleaseAck
-----------------------------------------------
```

```
Constants:
[0, 1]
-----------------------------------------------
```

G SUT implementation

```
package cav;

import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpRequest.BodyPublishers;
import java.net.http.HttpResponse.BodyHandlers;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Sut implements sut.interfaces.SutInterface {
    // create a client
    HttpClient client = HttpClient.newHttpClient();

    private Random random = new Random(1234567890);

    // Get fresh value
    private int getRandomInt() {
        return random.nextInt(10000000)+10000000;
    }

    // handling each Input

    /* register an uid
     */
    public OutputAction IRegister(int uid) throws IOException, InterruptedException {
        String methodName = "ONOK";
        List < Parameter > params = new ArrayList < Parameter > ();

        String reqBody = "{\"uid\":" + uid + "}";
```

```
    // create a request
    var request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:5000/register"))
        .header("Content-type", "application/json")
        .POST(BodyPublishers.ofString(reqBody))
        .build();

    // use the client to send the request
    var response = client.send(request, BodyHandlers.ofString());

    if (!response.body().trim().contains("failed")) {
        var pwd = Integer.parseInt(response.body().trim());
     methodName = "OOK";

        params.add(new Parameter(pwd, 0));
    }

    return new OutputAction(methodName, params);
}

/* login an user with uid
 */
public OutputAction ILogin(int uid,int pwd) throws IOException, InterruptedException {
    String methodName = "ONOK";
    List < Parameter > params = new ArrayList < Parameter > ();

    String reqBody = "{\"uid\":" + uid + ", \"pwd\":" + pwd + "}";

    // create a request
    var request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:5000/login"))
        .header("Content-type", "application/json")
        .POST(BodyPublishers.ofString(reqBody))
        .build();

    // use the client to send the request
    var response = client.send(request, BodyHandlers.ofString());
```

```
    var result = Integer.parseInt(response.body().trim());

    if (result == 1) {
     methodName = "OOK";
        params.add(new Parameter(getRandomInt(), 0)); // none important fresh output
    }

    return new OutputAction(methodName, params);
}

/* ILogout
 */
public OutputAction ILogout(int uid) throws IOException, InterruptedException {
    String methodName = "ONOK";
    List < Parameter > params = new ArrayList < Parameter > ();

    String reqBody = "{\"uid\":" + uid + "}";

    // create a request
    var request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:5000/logout"))
        .header("Content-type", "application/json")
        .POST(BodyPublishers.ofString(reqBody))
        .build();

    // use the client to send the request
    var response = client.send(request, BodyHandlers.ofString());
    var result = Integer.parseInt(response.body().trim());

    if (result == 1) {
     methodName = "OOK";
        params.add(new Parameter(getRandomInt(), 0)); // none important fresh output
    }

    return new OutputAction(methodName, params);
}
```

```
/* IChangePassword
 */
public OutputAction IChangePassword(int uid) throws IOException, InterruptedException {
    String methodName = "ONOK";
    List < Parameter > params = new ArrayList < Parameter > ();

    String reqBody = "{\"uid\":" + uid + "}";

    // create a request
    var request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:5000/change-password"))
        .header("Content-type", "application/json")
        .POST(BodyPublishers.ofString(reqBody))
        .build();

    // use the client to send the request
    var response = client.send(request, BodyHandlers.ofString());

    if (!response.body().trim().contains("failed")) {
        var pwd = Integer.parseInt(response.body().trim());
     methodName = "OOK";
        params.add(new Parameter(pwd, 0));
    }

    return new OutputAction(methodName, params);
}


// handling all inputs
public OutputAction handle(InputAction inputAction) throws IOException, InterruptedExce
    String methodName=inputAction.getMethodName();
    if (methodName.equals("ILogout")) {
        List < Parameter > params = inputAction.getParameters();
        int uid=params.get(0).getValue();
        return ILogout(uid);
    } else if (methodName.equals("IRegister")) {
        List < Parameter > params = inputAction.getParameters();
        int uid=params.get(0).getValue();
```

```
            return IRegister(uid);
        } else if (methodName.equals("IChangePassword")) {
            List < Parameter > params = inputAction.getParameters();
            int uid=params.get(0).getValue();
            return IChangePassword(uid);
        } else if (methodName.equals("ILogin")) {
            List < Parameter > params = inputAction.getParameters();
            int uid=params.get(0).getValue();
            int pwd=params.get(1).getValue();
            return ILogin(uid,pwd);
        }

        throw new RuntimeException("SUT does not support the input:" + inputAction.getMet
    }

   @Override
public sut.interfaces.OutputAction sendInput(sut.interfaces.InputAction origInputAction)
InputAction inputAction = new InputAction(origInputAction); // make copy to be safe!!
OutputAction outputAction;
        try {
            outputAction = handle(inputAction);
            return outputAction; // outputAction implements sut.interfaces.OutputAction i
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        throw new RuntimeException("Something went wrong");
}

@Override
public void sendReset() {
        // create a request
        var request = HttpRequest.newBuilder()
            .uri(URI.create("http://127.0.0.1:5000/reset"))
```

```
            .header("Content-type", "application/json")
            .POST(BodyPublishers.ofString(""))
            .build();

        // use the client to send the request
        try {
            client.send(request, BodyHandlers.ofString());
        } catch (IOException | InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}

}
```

H Contents of config.yaml of custom SUT

```
learning:
    mode: "stateVarN"
    sutinfoFile: "sutinfo.yaml"
    useExistingJarfile: "sut.jar"

    # no counterexample can be found
    seed: 129977735602
    # useSutSimulation: true
    reuseCounterExample: true
    # memVOrder: positionalHistory
    reconstructAbstractions: true
    reductionStrategies: ["loop","singleTransition"]
    maxTime: 3000
testing:
   # methods: [randomWalk]
    minValue: 0
    maxValue: 400
    maxTraceLength: 100
    maxNumTraces: 20000
verification:
```

```
    method: traces
    verificationTraces:
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> ILogin(2,-11) -> ILogin(1,-10)
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> IRegister(4) -> ILogin(3,-12) -
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> IRegister(4) -> IRegister(5) ->
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> IRegister(4) -> IRegister(5) ->
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> ILogin(1,-10) -> IChangePasswor
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> ILogin(2,-10) -> IChangePasswor
        - IRegister(1) -> IRegister(2) -> IRegister(3) -> ILogin(3,-10) -> IChangePasswor
learnResults:
    outputDir: "output/${sutname}"
sutInterface:
    directCall:
       package: "cav"
       className: "Sut"
sutImplementation:
    modelFile: model.uppaal.xml
    modelFilePath: input/Do_FreshSimpleOutput/model.uppaal.xml
    modelName: ${modelDirName}
    projectClassPath: build
logging:
    childLoggerLevels : {
        #abslearning.tree.trace.LookaheadSutTraceStateVar: "off",
    }
```

I API

```
from flask import Flask
from flask_restful import Resource, Api, reqparse
import ast, random

app = Flask(__name__)
api = Api(app)

# Initialize statemachine constants,variables and locations

id2pwd = dict()
```

```
id2loggedin = dict()
loggedin_users = 0

MAX_REGISTERED_USERS = 2
MAX_LOGGEDIN_USERS = 1000000


# Endpoints

class Register(Resource):
    def post(self):
        global MAX_REGISTERED_USERS
        global id2loggedin
        global id2pwd

        parser = reqparse.RequestParser() # initialize
        parser.add_argument('uid', required=True)  # add args
        args = parser.parse_args()
        uid = int(args['uid'])

        if ((not (uid in id2pwd)) and len(id2pwd) < MAX_REGISTERED_USERS):
            pwd = random.randint(100000, 1000000)
            id2pwd[uid] = pwd
            id2loggedin[uid] = False;

            return pwd, 200
        else:
            return "failed", 200

class Login(Resource):
    def post(self):
        global loggedin_users
        global MAX_LOGGEDIN_USERS
        global id2loggedin
        global id2pwd

        parser = reqparse.RequestParser() # initialize
        parser.add_argument('uid', required=True)
```

```
        parser.add_argument('pwd', required=True)
        args = parser.parse_args()
        uid = int(args['uid'])
        pwd = int(args['pwd'])

        if ((uid in id2pwd) and (not id2loggedin[uid]) and pwd == id2pwd[uid] and loggedi
            loggedin_users = loggedin_users + 1
            id2loggedin[uid] = True

            return 1, 200
        else:
            return 0, 200

class Logout(Resource):
    def post(self):
        global id2loggedin
        global loggedin_users

        parser = reqparse.RequestParser() # initialize
        parser.add_argument('uid', required=True)  # add args
        args = parser.parse_args()
        uid = int(args['uid'])

        if ((uid in id2loggedin) and id2loggedin[uid]):
            id2loggedin[uid] = False
            loggedin_users = loggedin_users - 1
            return 1, 200
        else:
            return 0, 200

class ChangePassword(Resource):
    def post(self):
        global id2loggedin
        global id2pwd

        parser = reqparse.RequestParser() # initialize
        parser.add_argument('uid', required=True)  # add args
        args = parser.parse_args()
```

```
        uid = int(args['uid'])

        if ((uid in id2loggedin) and id2loggedin[uid]):
            pwd = random.randint(100000, 1000000)
            id2pwd[uid] = pwd
            return pwd, 200
        else:
            return "failed", 200

class Reset(Resource):
    def post(self):
        global id2pwd
        global id2loggedin
        global loggedin_users

        id2pwd = dict()
        id2loggedin = dict()
        loggedin_users = 0

api.add_resource(Register, '/register')
api.add_resource(Login, '/login')
api.add_resource(Logout, '/logout')
api.add_resource(ChangePassword, '/change-password')
api.add_resource(Reset, '/reset')


# Run server
if __name__ == '__main__':
    app.run(debug = False)  # run our Flask app
```