Demonstrating Learning of Register Automata[*]

Maik Merten[1], Falk Howar[1], Bernhard Steffen[1], Sofia Cassel[2], and Bengt Jonsson[2]

[1] Technical University Dortmund, Chair for Programming Systems, Dortmund, D-44227, Germany
{maik.merten|falk.howar|steffen}@cs.tu-dortmund.de
[2] Dept. of Information Technology, Uppsala University, Sweden
{sofia.cassel|bengt.jonsson}@it.uu.se

Abstract We will demonstrate the impact of the integration of our most recently developed learning technology for inferring Register Automata into the LearnLib, our framework for active automata learning. This will not only illustrate the unique power of Register Automata, which allows one to faithfully model data independent systems, but also the ease of enhancing the LearnLib with new functionality.

1 Introduction

Active automata learning (aka regular extrapolation) has been proposed to semi-automatically infer formal behavioral models of underspecified systems. The resulting formal behavioral models can be used, e.g., for documentation or regression testing and thus can be an enabler for continued system evolution. Automated mediation between networked systems by automatically synthesizing connectors from behavioral models is a current research interest. This approach is currently in development in the CONNECT project [6].

To cater the various use-cases of automata-learning, LearnLib has been created to offer a versatile library of learning algorithms and related tools. Result of an extensive reengineering effort, the *Next Generation LearnLib* [8] implements a flexible component-based approach that supports quick iteration and refinement of learning setups (in the following the Next Generation LearnLib will simply be referred to as "LearnLib").

The reengineered LearnLib has seen continued evolution, for which a complete account will not be provided in this paper. We will rather focus on two main innovations:

– The modeling paradigm in LearnLib Studio underwent significant changes. For example, in the old modeling paradigm a dedicated setup phase would precede the actual learning process. This has been replaced by on-the-fly configuration during the learning phase itself, with only minimal static configuration needed beforehand.

[*] This work is supported by the European FP 7 project CONNECT (IST 231167).

– LearnLib has been outfitted with support for the Register Automata automaton model [4], which is a simple extension of finite automata with data from infinite domains. It can model data-independent systems [7], i.e., systems that do not compute or manipulate data but manage their adequate distribution, such as protocols and mediators, in an intuitive way.

The demonstration will not only cover these two points in isolation, but will also highlight the ease of integration of new functionality into the overall learning framework.

In the remainder of the paper, we will recall the basics of active automata learning in Section 2, followed by an introduction to Register Automata in Section 3. A learning solution for Register Automata will be presented in Section 4, demonstrating the innovations outlined above. In Section 5 a conclusion is provided, with references showcasing the broad application scope of the presented tool environment.

2 Active automata learning

Angluin's seminal algorithm L^* [2] defines two query types to gather information about the System Under Learning (SUL):

– Membership Queries (MQs) are traces of symbols from a predefined alphabet of inputs of the SUL. The learning algorithm will construct such input traces, execute these on the SUL and capture system output. From the gathered information a hypothesis model is generated.
– Equivalence Queries (EQs) compare the produced hypotheses with the target system. If the model is not accurate, a counterexample will be provided revealing a difference between the current hypothesis and the SUL. Evaluating counterexamples the learning algorithm will produce refined hypothesis models using additional MQs. Once no counterexample can be produced the learning procedure has produced an accurate model and can be stopped.

With those two query types, L^* is guaranteed to produce a minimal and correct model. In current practice, however, EQs can only be implemented approximately for a large class of systems, e.g., with additional invocations of the target system.

The original L^* algorithm has originally been presented for DFAs, but has since been adapted to Mealy Machines, which are a better fit for learning actual reactive systems as they can encode system output in a natural way. A major and recent increase in expressiveness is achieved with Register Automata [5], which are described in the following section.

3 Register Automata

Register Automata are an extension of finite automata with data from infinite domains and are, e.g., well-suited for describing communication protocols. Register Automata are defined as follows:

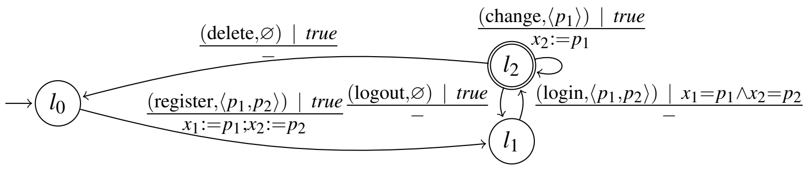


Figure 1. Partial RA model for a fragment of XMPP

Definition 1. *Let a* symbolic input *be a pair* $(a, \bar{p})$*, of a parameterized input a of arity k and a sequence of symbolic parameters* $\bar{p} = \langle p_1, \ldots, p_k \rangle$ *Let further* $X = \langle x_1, \ldots, x_m \rangle$ *be a finite set of* registers*. A* guard *is a conjunction of equalities and negated equalities, e.g.,* $p_i \neq x_j$*, over formal parameters and registers. An* assignment *is a partial mapping* $\rho : X \rightarrow X \cup P$ *for a set P of formal parameters.*

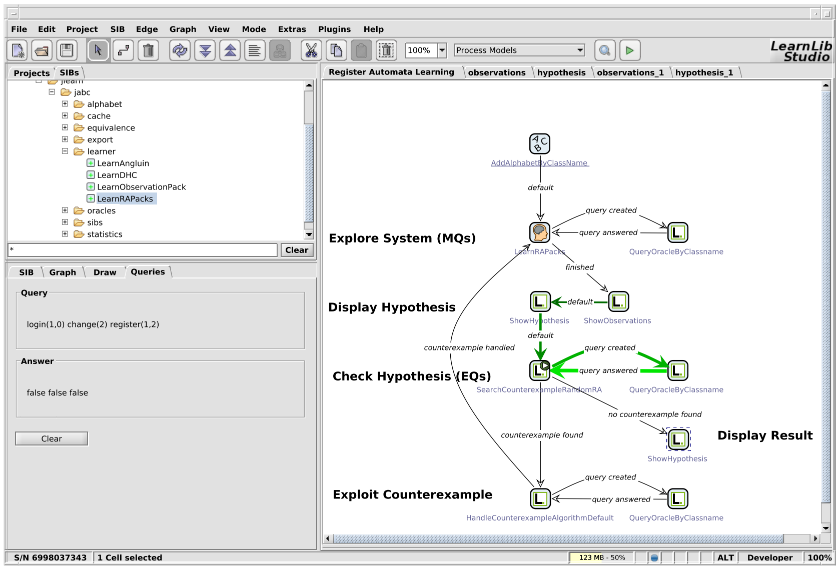
Definition 2. A *Register Automaton* (RA) is a tuple $\mathcal{A} = (A, L, l_0, X, \Gamma, \lambda)$, where

- A is a finite set of *actions.*
- L is a finite set of *locations.*
- $l_0 \in L$ is the *initial location.*
- X is a finite set of *registers.*
- Γ is a finite set of *transitions*, each of which is of form $\langle l, (a, \bar{p}), g, \rho, l' \rangle$, where l is the *source location*, l' is the *target location*, $(a, \bar{p})$ is a parameterized action, g is a guard, and ρ is an assignment.
- $\lambda : L \mapsto \{+, -\}$ maps each location to either $+$ (accept) or $-$ (reject). □

Let us define the semantics of an RA $\mathcal{A} = (A, L, l_0, X, \Gamma, \lambda)$. A X-valuation, denoted by ν, is a (partial) mapping from X to D. A *state* of $\mathcal{A}$ is a pair $\langle l, \nu \rangle$ where $l \in L$ and ν is a X-valuation. The *initial state* is $\langle l_0, \nu_0 \rangle$, i.e., the pair of initial location and empty valuation.

A *step* of $\mathcal{A}$, denoted by $\langle l, \nu \rangle \xrightarrow{(a, \bar{d})} \langle l', \nu' \rangle$, transfers $\mathcal{A}$ from $\langle l, \nu \rangle$ to $\langle l', \nu' \rangle$ on input $(a, \bar{d})$ if there is a transition $\langle l, (a, \bar{p}), g, \rho, l' \rangle \in \Gamma$ such that (1) g is modeled by $\bar{d}$ and ν, i.e., if it becomes true when replacing all p_i by d_i and all x_i by $\nu(x_i)$, and such that (2) ν' is the updated X-valuation, where $\nu'(x_i) = \nu(x_j)$ wherever $\rho(x_i) = x_j$, and $\nu'(x_i) = d_j$ wherever $\rho(x_i) = p_j$.

An example instance of a Register Automaton is provided in Figure 1, which models a subset of the XMPP instant messaging protocol focused on aspects of user authentication. In location l_0 no user account exists. With the parameterized action *register* a new account can be created, with the parameters p_1 and p_2 denoting a username and password. When executing the *register* action, the parameter values are copied into the registers x_1 and x_2 respectively. This action is unconditionally invocable in l_0, meaning that its guard is *true*. In contrast, the *login* in action of l_1 has a guard that specifies that the parameters p_1 and p_2 provided with the login action have to match the register contents of x_1 and x_2, i.e., the credentials provided during login have to match the ones stored in the registers. The other system actions represent logging out, changing the account password, and deleting the user account.



Query

login(1,0) change(2) register(1,2)

Answer

false false false

Clear

Figure 2. A modeled learning setup created in LearnLib Studio. The model is currently executed, with bold edges denoting the path of execution. The current query and its answer are made visible in the panel on the lower left side.

4 The tool demo

We will present the LearnLib framework and highlight recently integrated innovations. LearnLib Studio allows the model-based composition and execution of learning setups, where LearnLib components are made available as reusable building blocks. In Figure 2 a learning setup tooled for learning register automata is shown, configured to learn the system shown in Figure 1. The result of executing this learning setup is presented in Figure 3.

The reworked modeling approach: In the learning setup presented in Figure 2 the distinct pattern specific to active automata learning can be witnessed: a learning algorithm is invoked, resulting in MQs an "oracle" has to answer. In this context, oracles are components that execute queries on a target system and gather the invocation results, meaning they produce an answer for a given query. Once the learning algorithm has formed a hypothesis it can be displayed and the data structure gathering the observations can be visualized, e.g., for debugging purposes. In the displayed setup, EQs are approximated by a random walk conformance test, which generates additional queries that are answered by the system oracle. If no counterexample can be found, the learning procedure will terminate, displaying the final result. Otherwise the counterexample will be consumed and analyzed, which can result in the production of additional queries. Once these have been answered and the counterexample is exploited so that a

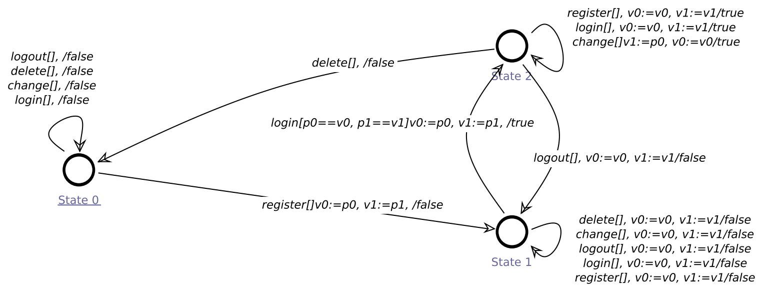

register[], v0:=v0, v1:=v1/true
login[], v0:=v0, v1:=v1/true
change[]v1:=p0, v0:=v0/true
logout[], /false
delete[], /false
change[], /false
login[], /false
delete[], /false
State 2
login[p0==v0, p1==v1]v0:=p0, v1:=p1, /true
logout[], v0:=v0, v1:=v1/false
State 0
register[]v0:=p0, v1:=p1, /false
delete[], v0:=v0, v1:=v1/false
change[], v0:=v0, v1:=v1/false
logout[], v0:=v0, v1:=v1/false
login[], v0:=v0, v1:=v1/false
register[], v0:=v0, v1:=v1/false
State 1

Figure 3. The resulting model from executing the learning setup. Guards are in square brackets, the next register contents are explicitly denoted on every transition as is the acceptance status of the following state. Register values are denoted as v_i.

refined hypothesis can be produced, the learning process will restart the learning algorithm.

The general workflow is reusable in nature. The only application-specific parts that have to be provided are the definition of the alphabet and the system oracle, which has to interface the SUL. These application-specific parts can be loaded in a standardized way with parameterized building blocks.

Compared to the previous modeling style supported by LearnLib Studio, the new modeling style is much improved in terms of usability and flexibility: in the old modeling paradigm, a dedicated setup phase would create a fixed configuration, connecting, e.g., the learning algorithm to the SUL oracle. This setup would then be instantiated and started, with limited ways to influence the behavior of the setup afterwards. In the new modeling paradigm, only an alphabet has to be specified beforehand, after which the learning algorithm can directly begin operation. Learning queries are delegated on-the-fly according to the setup execution flow. This greatly increases flexibility, as, for instance, setups can decide at runtime what oracle instances are used to answer queries.

Integration of Register Automata learning: Thanks to the flexible component-based approach of LearnLib, components for learning this new model type could be integrated into the general framework without changes to the LearnLib architecture.

In the learning setup presented above, the only components that are specific for the RA machine model are the building blocks encapsulating the learning algorithm and the equivalence approximation. The overall infrastructure provided by the LearnLib is reused to a high degree when using RAs despite adapting a much richer automata model. The overall flavor of how learning setups can be created is completely unchanged compared to how setups are specified for other machine models, abstracting from the details of the underlying data structures and algorithms and thus shielding the user from additional complexity.

Due to being integrated into the component framework, learning setups for Register Automata can use all facilities LearnLib offers for debugging, visualization and statistics. Thus our extension provides a powerful and unique framework for learning data independent systems [7].

5 Conclusion

The LearnLib framework and accompanying tools provide a rich environment for experimentation with the new Register Automata formalism. Embedded into a flexible component model, much functionality is shared between learning setups for different machine models, which enables a high degree of reuse. Users already versed in the operation of the LearnLib Studio will not have to learn a new style of modeling when adapting Register Automata, while users new to LearnLib Studio are only exposed to a limited set of generic concepts that are easy to understand.

The LearnLib is mature and used by several independent research groups. It has, e.g., been used to infer the behavior of a electronic passports [1], in security research [3], and it is a central enabler within the CONNECT framework.

LearnLib is available for download at `http://www.learnlib.de` and free for all academic purposes.

References

1. Fides Aarts, Julien Schmaltz, and Frits W. Vaandrager. Inference and Abstraction of the Biometric Passport. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (1)*, volume 6415 of *Lecture Notes in Computer Science*, pages 673–686. Springer, 2010.
2. Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.
3. G. Bossert, G. Hiet, and T. Henin. Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion Detection Systems. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1 –8, may 2011.
4. Sofia Cassel, Falk Howar, Bengt Jonsson, Maik Merten, and Bernhard Steffen. A Succinct Canonical Register Automaton Model. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 366–380. Springer Berlin / Heidelberg, 2011.
5. Falk Howar, Bernhard Steffen, Sofia Cassel, and Bengt Jonsson. Inferring Canonical Register Automata. In *VMCAI 2012, to appear.*
6. Valérie Issarny, Bernhard Steffen, Bengt Jonsson, Gordon S. Blair, Paul Grace, Marta Z. Kwiatkowska, Radu Calinescu, Paola Inverardi, Massimo Tivoli, Antonia Bertolino, and Antonino Sabetta. CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In *ICECCS*, pages 154–161, 2009.
7. R. Lazic and D. Nowak. A unifying approach to data-independence. In C. Palamidessi, editor, *Proc. CONCUR 2000, 11th Int. Conf. on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 581–595, 2000.
8. Maik Merten, Bernhard Steffen, Falk Howar, and Tiziana Margaria. Next generation learnlib. TACAS'11/ETAPS'11, pages 220–223, Berlin, Heidelberg, 2011. Springer-Verlag.