Next Generation LearnLib*

Maik Merten[1], Bernhard Steffen[1], Falk Howar[1], and Tiziana Margaria[2]

[1] Technical University Dortmund, Chair for Programming Systems, Dortmund, D-44227, Germany
`{maik.merten|steffen|falk.howar}@cs.tu-dortmund.de`

[2] University Potsdam, Chair for Service and Software Engineering, Potsdam, D-14482, Germany
`tiziana.margaria@cs.uni-potsdam.de`

Abstract The Next Generation LearnLib (NGLL) is a framework for model-based construction of dedicated learning solutions on the basis of extensible component libraries, which comprise various methods and tools to deal with realistic systems including test harnesses, reset mechanisms and abstraction/refinement techniques. Its construction style allows application experts to control, adapt, and evaluate complex learning processes with minimal programming expertise.

1 Introduction

Creating behavioral models of un(der)specified systems, e.g., for documentation- or verification-purposes, using (semi-)automated learning algorithms, has become a viable method for quality assurance. Its practical impact increases with the advances in computer resources and, in particular, with the ability to exploit application-specific frame conditions for optimization. Still, creating fitting learning setups is laborious, in part because available learning methods in practice are not engineered to be versatile, often being hard-coded for specific use cases and thus showing limited potential for adaptability towards new fields of application. The Next Generation LearnLib (NGLL) is designed to ease this task by offering an extensive and extensible component library comprising various methods and tools to deal with realistic systems including test harnesses, reset mechanisms and abstraction/refinement techniques. A modeling layer based on the NGLL allows for model-based construction of easily refinable learning solutions. Being internet-enabled, NGLL supports the integration of remote components. Thus learning solutions can be composed of mixtures of components running locally or anywhere in the world, a fact that can in particular be exploited to learn remote systems or to flexibly distribute the learning effort on distributed resources.

In the remainder of this paper, we will describe the technology underlying the NGLL in Section 2, present model-driven creation of learning setups in Section 3, and outline the usefulness in a competitive environment in Section 4, before we conclude in Section 5.

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

2 Base Technology

The NGLL is the result of an extensive reengineering effort on the original LearnLib [7], which has originally been designed to systematically build finite state machine models of unknown real world systems (Telecommunications Systems, Web Services, etc.). The experience with the LearnLib soon led to the construction of a platform for experimentation with different learning algorithms and to statistically analyze their characteristics in terms of learning effort, run time and memory consumption. The underlying learning technology is active learning following the pattern of Angluin's L^* algorithm [2], which introduced active system interrogation to automata learning. One of the main obstacles in practical learning is the implementation of the idealized form of interrogation in terms of membership and equivalence queries proposed by Angluin. This requires an application-specific interplay of testing and abstraction technology, driving the reengineering effort that created the NGLL.

The foundation of NGLL is a new extensive Java framework of data structures and utilities, based on a set of interface agreements extensively covering concerns of active learning from constructing alphabets to tethering target systems. This supports the development of new learning components with little boilerplate code and the integration of third-party learning technology, such as libalf [3].

All learning solutions we know of, like libalf, focus on providing fixed sets of learning algorithms. In contrast, the component model of the NGLL extends into the core of the learning algorithms, enabling application-fit tailoring of learning algorithms, at design- as well as at runtime. In particular, it is unique in

– comprising features for addressing real-world or legacy systems, like instrumentation, abstraction, and resetting,
– resolving abstraction-based non-determinism by alphabet abstraction refinement, which would otherwise lead to the failure of learning attempts [4],
– supporting execution and systematic experimentation and evaluation, even including remote learning and evaluation components, and, most notably, in
– its high-level modeling approach described in the next section.

3 Modeling Learning Solutions

LearnLib Studio, which is based on jABC [9], our service-oriented framework for the modeling, development, and execution of complex applications and processes, is NGLL's graphical interface for designing and executing learning and experimentation setups.

A complete learning solution is usually composed of several components, some of which are optional: learning algorithms for various model types, system adapters, query filters and caches, model exporters, statistical probes, abstraction providers, handlers for counterexamples etc.. Many of these components are reusable in nature. NGLL makes them available as easy-to-use building blocks for the graphical composition of application-fit learning experiments.

Figure 1 illustrates the graphical modeling style typical for LearnLib Studio along a very basic learning scenario. One easily identifies a common three phase

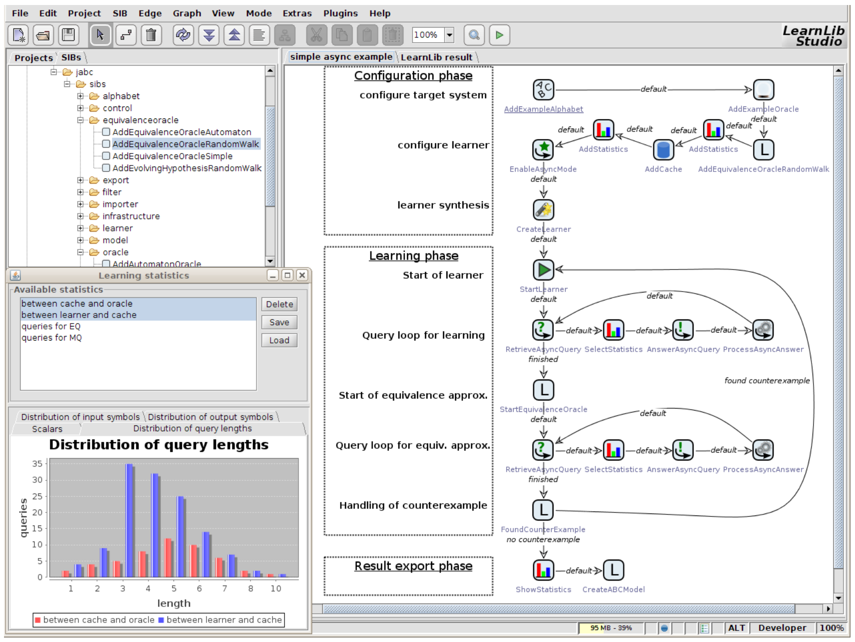

Figure 1. Executable model of a simple learning experiment in LearnLib Studio.

pattern recurring in most learning solutions: The learning process starts with a configuration phase, where in particular the considered alphabet and the system connector are selected, before the learner itself is created and started. The subsequent central learning phase is characterized by the L^*-typical iterations, which organize the test-based interrogation of the system to be learned. These iterations are structured in phases of exploration, which end with the construction of a hypothesis automaton, and the (approximate) realization of the so-called equivalence query, which in practice searches for counterexamples separating the hypothesis automaton from the system to be learned. If this search is successful, a new phase of exploration is started in order to take care of all the consequences implied by the counterexample. Otherwise the learning process terminates after some postprocessing in the third phase, e.g., to produce statistical data.

Most learning experiments follow this pattern, usually enriched by application-specific refinements. Our graphical modeling environment is designed for developing such kinds of refinements by supporting, e.g., component reuse, versioning, optimization and evaluation.

4 Fast-cycle experimentation: The ZULU Experience

The ability to quickly design and extend learning setups, coupled with statistical probes and visualizations, was invaluable during the ZULU competition [1]. Various learning setups, involving different learning algorithms and strategies for finding counterexamples, were evaluated in a time-saving manner in the graphical environment. The NGLL allows one to configure whole experimentation series for automatic evaluation in batch mode, resulting in aggregated statistical charts highlighting the various profiles. This way we were able to identify the winning setup for the ZULU competition, by playing with variants of finding and evaluating counterexamples [8,6] and combining them to a continuous evolution process for the construction of learning hypotheses [5].

5 Conclusion

The NGLL provides a machine learning framework, designed for flexibility, extensibility and reusability. It comprises LearnLib Studio, which enables quick experimentation with learning methods in research and practice, and thus helps to design fitting learning setups for application-specific contexts. Being executable jABC graphs, learning setups in LearnLib Studio can use every facility of the jABC-framework. This includes step-by-step execution, transformation of learning setups into standalone applications using code generation, parallel and hierarchical structuring of the models, model-checking, and automatic deployment on various platforms. Many concepts only briefly mentioned, but not discussed here in detail due to limited space, will be demonstrated during the tool demo. In experiments the NGLL demonstrated the ability to learn models with approximately 40,000 systems states and 50 alphabet symbols. The NGLL is available for download at the `http://www.learnlib.de` website.

References

1. Zulu - Active learning from queries competition, 2010. http://labh-curien.univ-st-etienne.fr/zulu/.
2. Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 2(75):87–106, 1987.
3. Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. libalf: The Automata Learning Framework. In *CAV'10*, pages 360–364, 2010.
4. Falk Howar, Bernhard Steffen, and Maik Merten. Automata Learning with Automated Alphabet Abstraction Refinement. In *Verification, Model Checking, and Abstract Interpretation, January 23-25, 2011, Austin, Texas, USA*.
5. Falk Howar, Bernhard Steffen, and Maik Merten. From ZULU to RERS - Lessons learned in the ZULU challenge. In *4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, volume 6415 of *Lecture Notes in Computer Science*. Springer, 2010.
6. Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, USA, 1994.
7. Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. LearnLib: a framework for extrapolating behavioral models. *Int. J. Softw. Tools Technol. Transf.*, 11(5):393–407, 2009.
8. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.
9. Bernhard Steffen, Tiziana Margaria, Ralf Nagel, Sven Jörges, and Christian Kubczak. Model-Driven Development with the jABC. In Eyal Bin, Avi Ziv, and Shmuel Ur, editors, *Haifa Verification Conference*, volume 4383 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2006.