Incremental Inference of Black-Box Components to support Integration Testing

Muzammil Shahbaz
France Telecom R&D
BP 98, 38243 Meylan Cedex, France
muhammad.muzammilshahbaz@francetelecom.com

Abstract

Model Based Testing relies on the availability of formal models that are indispensable in analyzing the complete system's behavior and testing of the key functionalities. On the other hand, the system designers of the industry are mostly relying on the integration of readymade software components (COTS) to build complex applications e.g. telecom services. Unfortunately, they are not provided with formal models or with reasonable documentation. The spirit of the thesis is to devise techniques to build formal models of black-box components, and to adopt test-generation strategies based upon the learned models to support integration testing. As a first step, we are modifying previous learning method, i.e., Angluin's algorithm to work with (extended) FSM models, which incorporates input/output parameters and predicates as well. Our framework from telecom industry focuses on methods that will be applied to large-scale components. That's why we are gradually moving towards richer models such as EFSMs. [1]

1 Motivation

Most telecom services are now developed by integrating software components (COTS) from third party sources, and linking them with some interfacing and orchestration glue to provided the service. A typical service could integrate several access components (e.g. a Web portal, a mobile access, a voice access), a content server, charging and billing components etc. The designer of the integrated system must test the interactions between the components. In general, only a small part of the general purpose components is exercised in their integration. In order to identify relevant tests of the interactions, we would like to base our tests on formal models of the components. But COTS are usually not provided with complete and precise models that could help in integration testing. In practice, maintaining the formal models is unrealistic because COTS evolve over time that quickly invalidates the original design sketch. The need of formal models as a prerequisite in using COTS is a daunting prospect to the designers of large-scale systems. Mostly they rely on their own intuitions to steer the testing effort and to evaluate the test results.

A solution to the problem is to generate models directly from the components. Automatic generation of formal models remains an active research domain. Various static and dynamic approaches have been proposed to generate abstract models from given source code [9]. On the contrary, COTS come from third-party sources whose internal structure is usually unknown, and are regarded as black-box components. Therefore, model generation techniques from code cannot be applied in this case. The only way to learn these components is to interact with their external interfaces, where their sequence of inputs and outputs, namely behavior traces can be observed. These behavior traces will be helpful in building a model. This way of learning models is difficult and leaves open questions, e.g., i) How the complexity of components that can generate large behavioral traces can be handled? ii) How can it be sure that the learned model is a "reasonable" approximation of the unknown model? These questions are challenging in order to apply methods based upon machine observations.

Our goal is to devise techniques and algorithms to learn black box components whose formal models are not given. Once the components are learned, we need to develop integration testing techniques based upon their models. In this vein, we rely on previous methods for automatically deriving models using an iterative learning approach. We do not envision being able to deduce complete and correct models. As a matter of fact, one can never explore a complete behavioral spectrum of the system. We aim at providing partial (or approximate) models which will help system designers in better understanding of the system. We assume models will be learned as (extended) state machines so that integration testing can be based upon the standard techniques [7]. Our application domain is a telecom industry where

[1] This PhD project has been started in Nov. 2005 with a collaboration between France Telecom R&D and Institut National Polytechnique de Grenoble, France.

systems, e.g. call center solutions, web-based applications and other related services, are usually composed of different components. Figure 1 depicts an overview of the framework, in which several black box components are integrated and are communicating with each other. The tester interacts with the system with the help of component's external interfaces and observes their internal communication.

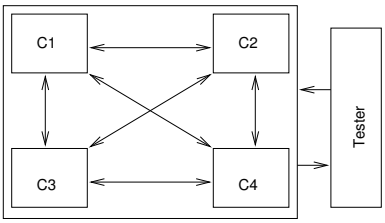

Figure 1. An overview of the framework. C1,C2,C3,C4 are black box software components.

2 Related Work

To develop learning methods for black box components, we start with the previous work on state machine inference. The problem has been well studied by the theoretical community (see Pitt's survey [10]). Gold [3] shows that finding a minimal deterministic finite automaton (DFA) of a black box machine is NP-Hard. Valiant [13] proposes PAC (Probably Approximately Correct) model for learning concepts from examples. The goal of PAC learning algorithm is to obtain DFA in polynomial time, with high probability, that is a good approximation of target DFA. Even approximate learnability of DFA was proven to be hard problem. The initial approaches were based on passive learning that relies mostly on given information. The research [11] shows that passively learning approach is apparently intractable, rather combination of active and passive learning is feasible. In active learning approach, the learning algorithm asks questions to eludicate conflicts in the model. In this framework, Angluin [1] proposes an algorithm that can infer a minimized Deterministic Finite Automaton (DFA) of an unknown machine in a polynomial time. This has been considered as a remarkable work and is applied in various domains. For example, Rivest & Schapire [12] adapt this algorithm in map learning scenarios. They improve the algorithm by removing its reset assumption with the help of homing sequences. Peled et al. [4] use this algorithm in Adaptive Model Checking. Since the last decade, this algorithm is being applied in somewhat more practical terms, i.e., to learn machines from real world examples. For example, Steffen et al. [6] studied domain specific optimizations to Angluin's Algorithm for prefix-closure, and have considered examples from telecom software systems. All this work considers real world examples, but does not address the issue of learning models other than DFA. In order to apply Angluin's Algorithm, they transform their initial models into DFAs. Steffen et al. [6] use I/O automata for their systems, but eventually transform them into DFAs. In our case, we consider telecommunication systems as complex reactive systems which receive signals from the environment, take decision on transitions, perform their inner computations and produce signals to the environment. In other words, a more adequate modeling of these systems yields indeed a finite automaton that incorporates inputs and outputs along with the parameters, and which do contain functions and predicates on such parameters. These models are well-suited for systems that contain very large input set. Existing algorithms for DFA cannot be used in learning such models, as the number of test queries can grow polynomially with the size of the input set. Also, these systems may have infinite parameter value domain. If some values are irrelevant or never used, the algorithm should ignore them. Thus, we need to develop further techniques which help in learning richer models. A very recent work [2] presents a modified version of Angluin algorithm to infer a parameterized system. The system contains predicates but does not take outputs and output parameters into account. Also it assumes that the parameters are of boolean type. Thus, we can conclude that the domain of learning richer models from black box machines is still not addressed in depth, and needs more contributions to cater its complexity.

3. Contributions

An overview of our expected contributions in the research is given below:

- We intend to deal with richer models which are more expressive than an ordinary automaton such as DFA. As a first step, we are extending Finite State Machines (FSM) with inputs and outputs along with the notions of parameters and predicates. We call this model as *Implicit Predicate Parameterized Finite State Machines* (PPFSM). For the time being, we are not addressing the issues of inferring functions, variables assignments and complex guards on transitions. They are not required in our integration testing approach. We believe that PPFSM models provide a good starting point which are not as simple as Label Transition Systems (LTS) and not as complicated as Extended Finite State Machines (EFSM). We can take benefit from previous works [5, 6, 2, 4] and can proceed further for learning richer models.

- We modify Angluin's algorithm to infer richer models

directly from the back box components. We do not use automata transformation techniques in order to apply original algorithm.

- We are addressing the problem of Integration Testing. Once the models are learned individually, they are composed to build an integrated system. At this point, we need an *Integration Testing Strategy* to derive test-cases from the learned models so that the integrated system of black box components could be tested in the vicinity of the explored behaviors.
- Original Angluin's algorithm assumes an oracle to obtain counterexamples when the learned models are incorrect. In our framework, we are using different techniques for getting a practical source of counterexamples. The idea is described in the section 4.

4. Sketch of the Approach

We intend to build an iterative approach for learning formal models of the black box components. A sketch of the approach is drawn in Figure 2. We start with testing black box machines individually with the help of a learning algorithm. The algorithm will generate systematic test cases that can be applied to the machine to observe its behaviors. We adapt this algorithm in such a way that it can learn richer models (e.g. PPFSM) directly from the observations. Once models are learned, we stop the first iteration and start composing models to build an integrated system (meaning, in that case, the integrated composition of models). The idea is that the input and output interfaces of the components are interconnected, so that an output interface of one component can be directed towards the input interface of the other component. This simplifies working of an integrated system in which components communicate by taking inputs from each other.

We say that the integrated system is well-defined if and only if i) no output is produced from any component in the communication that would be an unexpected input for the receiving components and ii) no deadlock or livelock is observed in the system. When any one of these conditions is violated, we trace the problem from the learned models and test the actual system of black box components with the help of that trace. In this case, the output of the actual system is considered as a counterexample, which will be supplied to the learning algorithm to refine the suspected models. This is the second iteration of the learning process and hence will continue until no counterexamples are found. Once the integrated system is well-defined, we test the actual system of black box components based upon our learned models. A *Test Generation Strategy* [8] can be adopted to generate test-cases from the learned models. Each time a test-case is executed, the actual outputs are compared to those provided by the models. This could reveal some contradictions, i.e., the output of the models do not conform to the output produced by the black box system. It terms as another source of counterexample, and thence the counterexample will again be provided to the learning algorithm to refine the suspected models. The iterative process goes on until no counterexample is found, or certain threshold limit is achieved on iterations, or certain coverage criteria is achieved on the models.

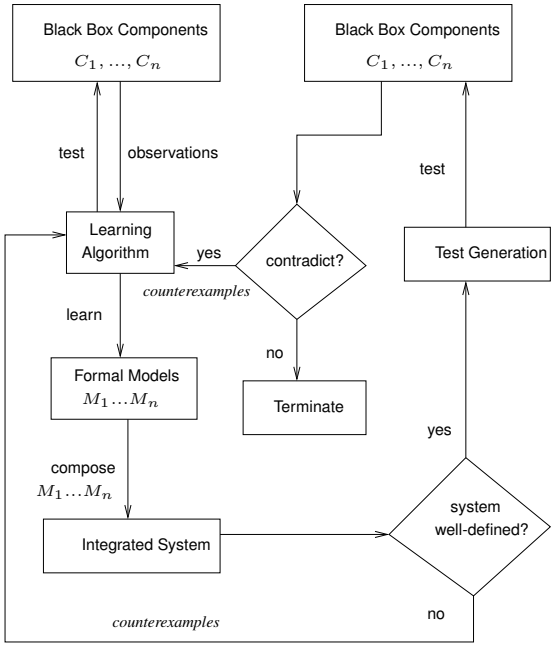

Figure 2. A sketch of the approach.

5. Example

We further illustrate our approach with the help of a small example. Consider a PPFSM model in the figure 3 as a black box component. The model is an extension of FSM with inputs and outputs, along with the parameters. The model has implicit predicates, e.g., the outward transitions of state 1 are labelled with the same input symbol a, but with different input parameter values 1 and 2 that take the transitions to different target states. We assume that the set of input types I and the set of parameter types D are known to learn the black box components. We base our

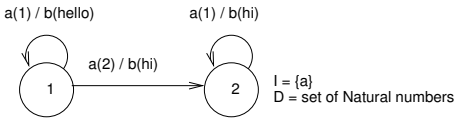

Figure 3. An Example of a PPFSM model.

algorithm on Angluin's to learn the PPFSM models of the components. The algorithm will generate systematic test-cases to observe the behaviors of the component. These observations are organized into two tables namely, *Primary Table (PT)* and *Auxiliary Table* (AT). The structure of the tables can be seen in the figure 4. The rows and columns are labelled by the elements of I [2]. The rows of PT records the output symbols produced by the component, whereas the rows of AT records the value pairs of input parameter values and their corresponding output parameter values. To simplify the example, we record only the last output symbol in PT and the last value pair in AT from the complete output sequence produced by the component. Two rows in PT are different if they contain different output symbols. Two rows in AT are different if for the same input parameter value they contain different output parameter values. Hence in the figure 4, the rows ϵ and a are different due to $(1, hello)$ and $(1, hi)$ recorded in AT. The row aa is similar to the row a because of the same outputs in both tables. When the execution of the possible test-cases is over, a PPFSM model of the component is conjectured. The different rows of the tables are regarded as different states of the PPFSM, and transitions are made with the help of the observations recorded in the tables. The conjecture made from the tables in the figure 4 is akin to the PPFSM model in the figure 3.

	a
ϵ	b
a	b
aa	b

	a
ϵ	(1,hello), (2,hi)
a	(1,hi), (2,hi)
aa	(1,hi)

Figure 4. PT (left) and AT (right) of the example in figure 3.

All black box components are learned in the same manner and finally their models are composed to build a well-defined integrated system. The next step is to devise an *Integration Testing Strategy* such that the designer can test the interesting behaviors of the complete black box system.

6. Conclusion

Our goal is to infer formal models of the black box components in isolation and then to test the integrated system of these components. At the same time, we are addressing the issue of learning rich models that can capture the peculiarities of the complex systems. We start with a parameterized model that is called PPFSM, and extend the Angluin's algorithm so that it can learn such models directly from the observations. We also try to find better solutions for obtaining counterexamples which are essential in refinement of the learned models. Later, we compose the learned models to build a well-defined integrated system, and then to perform integration testing with respect to these models.

We intend to explore these directions, and continue improving overall approach that leads towards the models, e.g. EFSMs, that are well-suited for the designing of complex systems.

[2] except the first row, which is labelled by ϵ, i.e., an empty string

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 2:87–106, 1987.

[2] T. Berg, B. Jonsson, and H. Raffelt. Regular inference for state machines with parameters. *Lecture Notes in Computer Science*, 3922:107–121, Mar. 2006.

[3] E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[4] A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 357–370, 2002.

[5] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In *Fundamental Approaches to Software Engineering*, pages 80–95, 2002.

[6] H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *CAV*, pages 315–327, 2003.

[7] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[8] K. Li, R. Groz, and M. Shahbaz. Integration testing of components guided by incremental state machine learning. In *TAIC PART*, Windsor, UK, 2006.

[9] J. W. Nimmer and M. D. Ernst. Automatic generation of program specifications. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, pages 229–239, New York, NY, USA, 2002. ACM Press.

[10] L. Pitt. Inductive inference, dfas, and computational complexity. In *AII '89: Proceedings of the International Workshop on Analogical and Inductive Inference*, pages 18–44, London, UK, 1989. Springer-Verlag.

[11] L. Pitt and M. K. Warmuth. The minimum consistent dfa problem cannot be approximated within and polynomial. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 421–432, New York, NY, USA, 1989. ACM Press.

[12] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. In *Machine Learning: From Theory to Applications*, pages 51–73, 1993.

[13] L. G. Valiant. A theory of the learnable. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445, New York, NY, USA, 1984. ACM Press.