Finding Counterexamples Fast: Lessons learned in the ZULU challenge

Falk Howar, Bernhard Steffen, Maik Merten

Finding Counterexamples Fast
Lessons learned in the ZULU challenge

Falk Howar, Bernhard Steffen, Maik Merten

University of Dortmund, Chair of Programming Systems,
Otto-Hahn-Str. 14, 44227 Dortmund, Germany
{falk.howar, steffen, maik.merten}@cs.tu-dortmund.de
Tel. ++49-231-755-7759, Fax. ++49-231-755-5802

Introduction

The concept of query learning (due to Dana Angluin [1]) is used and developed in several different communities today. The TU Dortmund team originates in the area of program verification. Active learning in this community is used to produce exact and complete models of a system under test. Having in mind this application of learning, we entered the ZULU challenge with the goal to improve in (1) saving membership queries and (2) finding counterexamples, while still producing exact models. In both disciplines we proceeded in two steps: first finding good generic solutions and then customizing these to reflect the specific profile of the ZULU problems. Our main contribution is a highly configurable learning algorithm, which can mimic most of the known algorithms, and a new approach to steering the search for counterexamples using a monotone growing hypothesis annotated with coverage information.

A Configurable Inference Framework

The learning algorithm we used can best be described as an implementation of generalized *Observation Packs* [2]. It combines a discrimination tree [3] with a reduced observation table [4]. The realization as a general framework allows us to switch between different strategies for handling counterexamples easily as well as using a non-uniform observation table (i.e., a table with multiple sets of distinguishing suffixes). Additionally, these sets can be initialized arbitrarily (e.g., as $\{\varepsilon\}$ for deterministic finite automata (DFA) or as Σ for Mealy machines).

For ZULU, we configured two versions of our learning algorithm, both using a strategy for analyzing counterexamples that is based on [4]. The strategy extracts from each counterexample (1) a new distinguishing suffix and (2) the word from the SA-set that will produce an unclosure subsequently. The registered algorithms differed as follows.

Initial set of distinguishing suffixes: In one configuration, the initial set of distinguishing suffixes was initialized as $\{\varepsilon\}$ (as in the literature). In the other configuration, we used $\{\varepsilon\} \cup \Sigma$ in order to simulate the effect of changing from DFA to Mealy models.

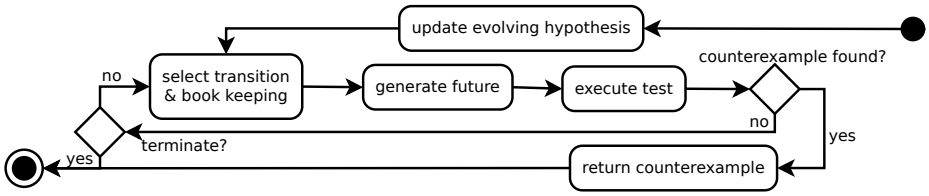

Figure 1. Continuous Equivalence Query

Observation Table: We used uniform and non-uniform observation tables. In a non-uniform table, the sets of distinguishing suffixes are managed independently for each component (cf. [2]). This leads to using less membership queries but more equivalence queries than a uniform table does.

Continuous Equivalence Queries

Active learning algorithms proceed in rounds. Each round is opened by an *equivalence query*. A counterexample returned from such a query will be analyzed and exploited during the rest of the round. This general formulation suggests that equivalence queries in different rounds are independent, i.e., run on unrelated conjectures. But, using a reduced observation table (or more precisely: analyzing counterexamples as in [4]) will guarantee a monotone growing hypothesis: the S-set of access sequences will only be extended by elements from the SA-set. The prefix-closed set S can be understood as a successively produced spanning tree of the target automaton. This observation has two consequences:

1. finding counterexamples coincides with proving transitions (elements from the set SA) leading to yet undiscovered states,
2. different conjectures can be related using the S-set.

Relating the different conjectures results in using only one *evolving* hypothesis automaton. In this automaton, only the transitions that correspond to elements from the SA-set can change (and only in a monotone fashion). This allows the formulation of continuous equivalence tests: each transition in the hypothesis can be annotated with a measure, e.g., counting the number of tests that have been executed for this transition; the counter will be reset when the transition changes. This global coverage criterion can be used to steer the search for counterexamples. Fig. 1 shows schematically one round of such a continuous equivalence query: after updating the hypothesis, single transitions are selected and then tested with some future (i.e., suffix word) until either a counterexample is found or a termination criterion is met. For the ZULU challenge, we concretized this general scheme as follows.

Select transition & Book keeping: For the *E.H.Blocking* algorithm, transitions from the SA-set were chosen randomly. Once used, a transition was excluded from subsequent tests. When no transitions were left to choose from,

Table 1. Algorithms: Configuration and Ranking

Algorithm	Dist. Set		Equivalence Query		Training	Rank
	Init.	Uniform	Continuous	Strategy	(Avg.)	
E.H.Blocking			yes	block transitions	89.38	1
E.H.Weighted	$\{\varepsilon\}$	no	yes	weight transitions	89.26	2
Random			no	random walks	88.93	6
run_random			no	random walks	80.17	14
run_blocking1	$\{\varepsilon\} \cup \Sigma$	yes	yes	block transitions	79.89	15
run_weighted1			yes	weight transitions	79.65	16

all transitions were re-enabled. The *E.H. Weighted* algorithm uses weights on all transitions, which will be increased each time a transition is selected. The probability of choosing a transition is inversely proportional to the weight.

Generate future: The suffixes were generated randomly with increasing length. The length was initialized as some ratio of the number of states in the hypothesis automaton, and was increased after a certain number of unsuccessful tests. The exact adjustment of the suffix length was developed in a trial-and-error way to fit the properties of the problems in the ZULU challenge.

We did not use an explicit termination criterion. A query terminated as soon as the number of queries granted by ZULU was reached.

Results

For the actual competition, we registered six candidate algorithms: three using a non-uniform observation table with a DFA-style initial set of distinguishing suffixes and three using a uniform observation table with a (modified) Mealy-style initial set of distinguishing suffixes. Those two groups correspond to the decisions discussed above.

Both groups were equipped with the same three equivalence algorithms: (1) E.H.Blocking, (2) E.H.Weighted, and (3) a plain random walks algorithm as reference. The random walks algorithm tested randomly generated words. Table 1 shows the configuration of the algorithms, their average scores during the training phase and the rankings from the competition phase.

On the training problems, we also ran algorithms for the other two possible configurations of the learning phase. There was, as can partly be seen in Table 1 and Table 2, a significant gap between algorithms using uniform tables and algorithms using non-uniform tables (about 10 to 15 points). For the algorithms using the same kind of table, there were gaps of 2 to 5 points between the versions with different initial sets of distinguishing suffixes; DFA-style initial sets leading to better results.

Compared to the big differences in the scores that were caused by the configuration of the learning phase, the differences between the three equivalence

Table 2. Detailed Training Example: Problem 49763507

Algorithm	New Membership Queries			Rounds	States	Score
	Close Obs.	Analyze CEs	Search CEs			
E.H.Blocking	6,744	358	999	259	352	94.11
E.H.Weighted	6,717	349	1,035	262	351	94.61
Random	6,586	519	996	228	332	93.28
run_random	8,080	14	7	5	312	74.89
run_blocking1	8,074	11	16	6	319	73.06
run_weighted1	8,077	9	15	6	319	74.39

algorithms at first glance seem to be not significant. But, this is due to the ZULU rating mechanism as shown in Table 2 for the training problem 49763507. While the ratings do not differ significantly, the number of states does. The continuous equivalence algorithms produce conjectures with significantly more states using the same amount of queries (over all) as the random walks algorithm does.

Conclusion

We played the ZULU challenge following the same pattern for the learning phase and for the equivalence phase. We first built good general frameworks and then customized these to reflect the special requirements of the ZULU scenario: The problems were best learned as DFA and using a non-uniform observation table. Both decisions aim at keeping the table as small as possible, i.e., saving membership queries. Counterexamples were found considerably faster (especially for bigger systems) using an evolving hypothesis. This allowed spending more queries on the learning part. Also, due to their structure, the analysis of the counterexamples from the continuous equivalence queries required less membership queries than analyzing randomly generated counterexamples.

The configuration we used for the learning phase made us compete at eye level with the other players. Finally, the solution for finding counterexamples fast resulted in the small but deciding advance (89.39 vs. 88.93 in the training phase, but 1st vs. 6th in the competition phase).

References

1. D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.
2. José L. Balcázar, Josep Díaz, and Ricard Gavaldà. Algorithms for Learning Finite Automata from Queries: A Unified View. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72, 1997.
3. Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.
4. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.