# An algorithm for learning real-time automata[1]

Sicco Verwer [a] Mathijs de Weerdt [a] Cees Witteveen [a]

[a] *Delft University of Technology, P.O. Box 5031, 2600 GA, Delft, the Netherlands*

**Introduction** A common model for discrete event systems is a *deterministic finite automaton* (DFA). An advantage of this model is that it can be interpreted by domain experts. When observing a real-world system, however, there often is more information than just the sequence of discrete events: the time at which these events occur may be very important. In such a case, the DFA model is too limited.

A variant of a DFA that includes the notion of time is called a *timed automaton* (TA). In this model, each symbol of a word occurs at a certain point in time. The execution of a TA depends not only on the type of symbol occurring, but also on the time that has elapsed since some previous symbol occurrence. We are interested in the problem of identifying such a time dependent system from a data sample.

**Real-time automata** In a TA, time values can be modeled using the natural numbers. A TA contains a finite number of synchronously increasing time values (clocks) and one boolean constraint on these time values for each transition (clock guards). A *real-time automaton* [1] is a TA with only one clock that represents the time delay between two consecutive events. The *delay guards* for the transitions are constraints on this time delay, represented by an interval in $\mathbb{N}$. We say that such a delay guard $G$ is *satisfied* by a time value $t \in \mathbb{N}$ if $t \in G$. A *real-time automaton* (RTA) is a tuple $\mathcal{A} = \langle Q, \Sigma, D, q_0, F \rangle$, where $Q$ is a finite set of states, $\Sigma$ is a finite set of symbols, $D$ is a finite set of transitions, $q_0$ is the start state, and $F \subseteq Q$ is a subset of final states. A transition $d \in D$ in this automaton is a tuple $\langle q, q', s, G \rangle$, where $q, q' \in Q$ are the source and target states, $s \in \Sigma$ is a symbol, and $G$ is a delay guard defined by an interval in $\mathbb{N}$.

We only regard deterministic (or unambiguous) RTAs. In an RTA it is not only possible to activate a transition to another state, but it is also allowed to remain in the same state for some time (delay). A transition to another state is possible only if its delay guard is satisfied by the current delay. Apart from this, an RTA computes strings in the normal DFA way. We try to identify an RTA from labeled examples by using a modified DFA identification algorithm.

**Learning real-time automata** Our algorithm for the identification of RTAs from a timed sample is based on the evidence driven state merging algorithm. State-merging identifies a DFA from a finite set of positive and negative examples. The idea of a state-merging algorithm is to first construct a tree automaton from this input, and then merge the states of this tree. In this tree there is a path from the root node to a node in the tree for each input sample. The node in which a positive or negative example ends is marked positive (accepting) or negative (rejecting), respectively. A *merge* of two states combines the states into one: all input transitions of both states point to this new node and this new node contains the output transitions of both nodes. Such a merge is only allowed if the states are *consistent*, i.e. when no positive node is merged with a negative node. Using an evidence measure, state merging is an efficient and good-performing DFA identification algorithm [2].

In addition to simply merging states, we need to deal with the time values and delay guards of the RTA identification problem. The key idea of our algorithm is to first assume that every delay

[1]Published in: Proceedings of the Annual Belgian-Dutch Machine Learning Conference (Benelearn) (pp. 128-135), 2007

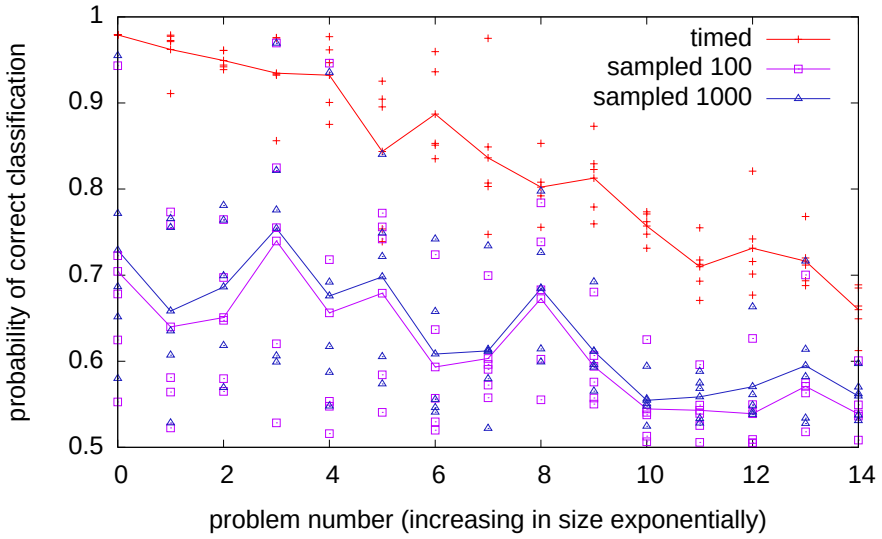


Figure 1: This graph shows the results obtained with a sample consisting of 2000 examples. The RTAs used to create these sets range from 2 states, with 1 split (nr. 1), to 32 states with 64 splits (nr. 14). Of each size we generated 5 different problem instances. The points show the probability of correct classification for each problem instance. The lines are the averages of these 5 results.

guard contains all delay values, and then to *split* them based on timed evidence. A *split* $s(d, t)$ of transition $d$ with clock guard $g = [t_1, t_2]$ at time $t$ divides $d$ into two new transitions $d'$ and $d''$, with delay guards $g' = [t_1, t]$ and $g'' = [t + 1, t_2]$ respectively.

We use a probability value as timed evidence. For each pair of input examples we determine the probability that they will end up in the same state if we were to choose split points uniformly at random. The evidence value increases if this probability is large for two examples with the same label. It decreases if this probability is large for two examples with a different label. This measure is intuitively appealing because it has less influence if the two examples have a larger time distance between them.

**Results** Our algorithm was tested on random data. We generated this data from a random RTA, with a fixed number of states, and a fixed number of split intervals. The split intervals are the result of applying the split routine at random time points to randomly picked transitions.

Our RTA learning algorithm is an alternative to the straightforward approach of first mapping the timed input sample to an untimed input sample using sampling, and then to learn a DFA from the untimed data using state merging. We sampled the timed data using fixed sampling lengths: 100 and 1000 time points. Figure 1 shows the result of our RTA learning algorithm compared to the sampling approach.

The RTA identification algorithm performs really well: it achieves 80% of correctly classified new examples for RTAs up to 16 states and 8 splits (number 9 in the graphs). This shows that it is possible to apply an algorithm such as ours to real-world problems.

# References

[1] Catalin Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6(1):3–23, 2001.

[2] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *ICGI*, volume 1433 of *LNCS*, pages 1–12. Springer, 1998.