A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data

Sicco Verwer[1], Mathijs de Weerdt[2], and Cees Witteveen[2]

[1] Eindhoven University of Technology
[2] Delft University of Technology
s.verwer@tue.nl,{M.M.deWeerdt,C.Witteveen}@tudelft.nl

Abstract. We adapt an algorithm (RTI) for identifying (learning) a *deterministic real-time automaton* (DRTA) to the setting of positive timed strings (or time-stamped event sequences). An DRTA can be seen as a deterministic finite state automaton (DFA) with time constraints. Because DRTAs model time using numbers, they can be exponentially more compact than equivalent DFA models that model time using states.
We use a new likelihood-ratio statistical test for checking consistency in the RTI algorithm. The result is the RTI+ algorithm, which stands for *real-time identification from positive data*. RTI+ is an efficient algorithm for identifying DRTAs from positive data. We show using artificial data that RTI+ is capable of identifying sufficiently large DRTAs in order to identify real-world real-time systems.

1 Introduction

In previous work [11], we described the RTI algorithm for identifying (learning) deterministic real-time automata (DRTAs) from labeled data, i.e., from an input sample $S = (S_+, S_-)$. The RTI algorithm is based on the currently best-performing algorithm for the identification of deterministic finite state automata (DFAs), called evidence-driven state-merging (ESDM) [9]. The only difference between DFAs and DRTAs are that DRTAs contain time constraints. In addition to using the standard state-merging techniques, RTI identifies these time constraints by splitting transitions into two, see [11] for details.

The RTI algorithm is efficient in both run-time and convergence because it is a special case of an efficient algorithm for identifying one-clock timed automata, see [12]. In practice, however, it can sometimes be difficult to apply RTI. The reason being that data can often only be obtained from actual observations of the process to be modeled. From such observations we only obtain timed strings that have actually been generated by the system. In other words, we only have access to the *positive data* S_+.

In this paper, we adapt the RTI algorithm to this setting. A straightforward way to do this is to make the model probabilistic, and to check for consistency using statistics. This has been done many times, and in different ways, for the

problem of identifying (probabilistic) DFAs, see, e.g., [2, 8, 3]. As far as we know, this is the first time such an approach is applied to the problem of identifying DRTAs.

We start this paper by defining DRTAs and *probabilistic DRTAs* (PDRTA, Section 2). In addition to a DRTA *structure*, a PDRTA contains *parameters* that model the probabilities of events in the DRTA structure. In order to identify a PDRTA, we thus need to solve two different identification problems: the first problem is to identify the correct DRTA structure, and the second is to set the probabilistic parameters of this model correctly. However, because a PDRTA is a deterministic model, we can simply set these parameters to the normalized frequency counts of events in the input sample S_+.[3] This is very easy to compute and it is the unique correct setting of the parameters given the data. We therefore focus on identifying the DRTA structure of a PDRTA.

We introduce a new likelihood-ratio test that can be used to solve this identification problem (Section 3). Intuitively, this test tests the null-hypothesis that the suffixes of strings that can occur after two different states have been reached follow the same PDRTA distribution, i.e., whether these two states can be modeled using a single state in a PDRTA. If this null-hypothesis is rejected with sufficient confidence, then this is considered to be evidence that these two states should not be merged. Equivalently, if these two states result from a split of a transition, then this is evidence that this transition should be split. In this way, the *statistical evidence* resulting from these tests replace the evidence value in the original RTI algorithm. The result is the RTI+ algorithm (Section 3.3), which stands for real-time identification from positive data. The RTI+ algorithm is an efficient algorithm for identifying DRTAs from positive data.

The likelihood-ratio test used by RTI+ is designed specifically for the purpose of identifying a PDRTA from positive data. Although many algorithms like RTI+ exist for the problem of identifying (probabilistic) DFAs, none of these algorithms uses the non-timed version of the likelihood-ratio test of RTI+. Hence, since this test can easily be modified in order to identify (probabilistic) DFAs, it also contributes to the current state-of-the-art in DFA identification.

In order to evaluate the performance of the RTI+ algorithm we show a typical result of RTI+ when run on data generated from a random PDRTA (Section 4). This result shows that our algorithm is capable of identifying sufficiently complex real-time systems in order to be useful in practice. We end this paper with some conclusions and pointers for future work (Section 5).

2 Probabilistic Deterministic Real-Time Automata

The following exposition uses basic notation from language, automata, and complexity theory. For an introduction the reader is referred to [10]. In the following, we first describe non-probabilistic real-time automata, then we show how to add probability distributions to these model.

[3] In the case of a non-deterministic model, setting the model parameters is a lot harder. In fact, it can be as difficult as identifying the model itself.

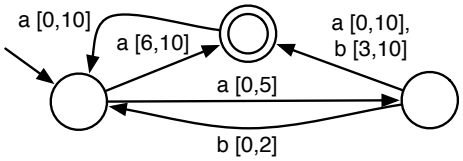

Fig. 1: An example of a DRTA. The leftmost state is the start state, indicated by the sourceless arrow. The topmost state is an end state, indicated by the double circle. Every state transition contains both a label a or b and a delay guard $[n, n']$. Missing transitions lead to a rejecting garbage state.

2.1 Real-Time Automata

In a real-time system, each occurrence of a symbol (event) is associated with a time value, i.e., its time of occurrence. We model these time values using the *natural numbers* $\mathbb{N}$. This is sufficient because in practice we always deal with a finite precision of time, e.g., milliseconds. Timed automata [1] can be used to accept or generate a sequence $\tau = (a_1, t_1)(a_2, t_2)(a_3, t_3)\dots(a_n, t_n)$ of symbols $a_i \in \Sigma$ paired with time values $t_i \in \mathbb{N}$, called a *timed string*. Every time value t_i in a timed string represents the time (delay) until the occurrence of symbol a_i since the occurrence of the previous symbol a_{i-1}.

In timed automata, timing conditions are added using a finite number of *clocks* and a *clock guard* for each transition. In this paper, we use a class of timed automata known as *real-time automata* (RTAs) [5]. An RTA has only one clock that represents the time delay between two *consecutive events*. The clock guards for the transitions are then constraints on this time delay. When trying to identify an RTA from data, one can always determine an upper bound on the possible time delays by taking the maximum observed delay in this data. Therefore, we represent a *delay guard* $[n, n']$ by a *closed interval* in $\mathbb{N}$.

Definition 1. *(RTA) A* real-time automaton *(RTA) is a 5-tuple* $A = \langle Q, \Sigma, \Delta, q_0, F\rangle$*, where* Q *is a finite set of states,* Σ *is a finite set of symbols,* Δ *is a finite set of transitions,* q_0 *is the start state, and* $F \subseteq Q$ *is a set of accepting states.*

A transition $\delta \in \Delta$ *in an RTA is a tuple* $\langle q, q', a, [n, n']\rangle$*, where* $q, q' \in Q$ *are the source and target states,* $a \in \Sigma$ *is a symbol, and* $[n, n']$ *is a delay guard.*

Due to the complexity of identifying non-deterministic automata (see [4]), we only consider *deterministic* RTAs (DRTAs). An RTA A is called *deterministic* if A does not contain two transitions with the same symbol, the same source state, and overlapping delay guards. Like timed automata, in DRTAs, it is possible to make time transitions in addition to the normal state transitions used in DFAs. In other words, during its execution a DRTA can remain in the same state for a while before it generates the next symbol. The time it spends in every state is represented by the time values of a timed string. In a DRTA, a state transition is possible (can fire) only if its delay guard contains the time spent in the previous state. A transition $\langle q, q', a, [n, n']\rangle$ of a DRTA is thus interpreted as follows:

whenever the automaton is in state q, reading a timed symbol (a, t) such that $t \in [n, n']$, then the DRTA will move to the next state q'.

Example 1. Figure 1 shows an example DRTA. This DRTA accepts and rejects timed strings not only based on their event symbols, but also based on their time values. For instance, it accepts $(a, 4)(b, 3)$ (state sequence: left $\rightarrow$ bottom $\rightarrow$ top) and $(a, 6)(a, 5)(a, 6)$ (left $\rightarrow$ top $\rightarrow$ left $\rightarrow$ top), and rejects $(a, 6)(b, 2)$ (left $\rightarrow$ top $\rightarrow$ reject) and $(a, 5)(a, 5)(a, 6)$ (left $\rightarrow$ bottom $\rightarrow$ top $\rightarrow$ left).

2.2 Adding Probability Distributions

In order to identify a DRTA from positive data S_+, we need to model a probability distribution for timed strings using a DRTA structure. Identifying a DRTA then consists of fitting this distribution and the model structure to the data available in S_+. We want to adapt RTI [11] to identify such *probabilistic DRTAs* (PDRTAs). Since they have the same structure as DRTAs, we only need to decide how to represent the probability of observing a certain timed event (a, t) given the current state q of the PDRTA, i.e., $Pr(O = (a, t) \mid q)$. In order to determine the probability distribution of this random variable O, we require two distributions for every state q of the PDRTA: one for the possible symbols $Pr(S = a \mid q)$, and one for the possible time values $Pr(T = t \mid q)$. The probability of the next state $Pr(X = q' \mid q)$ is determined by these two distributions because the PDRTA model is deterministic.

The distribution over events $Pr(S = a \mid q)$ that we use is the standard generalization of the Bernoulli distribution, i.e., every symbol a has some probability $Pr(S = a \mid q)$ given the current state q, and it holds that $\sum_{a \in \Sigma} Pr(S = a \mid q) = 1$ (also known as the multinomial distribution). This is the most straightforward choice for a distribution function and it is used in many probabilistic models, such as Markov chains, hidden Markov models, and probabilistic automata.

A flexible way to model a distribution over time $Pr(T = t \mid q)$ is by using histograms. A histogram divides the domain of the distribution (in our case time) into a fixed number of bins H. Every bin $[v, v'] \in H$ is an interval in $\mathbb{N}$. The distributions inside the bins are modeled uniformly, i.e., for all $[v, v'] \in H$ and all $t, t' \in [v, v']$, $Pr(T = t \mid q) = Pr(T = t' \mid q)$. Naturally, it has to hold that all these probabilities sum to one: $\sum_{t \in \mathbb{N}} Pr(T = t \mid q) = 1$. Using histograms to model the time distribution might look simple, but it is very effective. In fact, it is a common way to model time in hidden semi-Markov models, see, e.g., [6].

The price of using a histogram to model time is that we need to specify the amount, and the sizes (division points) $[v, v']$ of the histogram bins. Choosing these values boils down to making a tradeoff between the model complexity and the amount of data required to identify the model. More bins lead to a more complex model that is capable of modeling the time distribution more accurately, but it requires more data in order to do so. To simplify matters, we assume that these bins are specified beforehand, for example by a domain expert, or by performing data analysis.

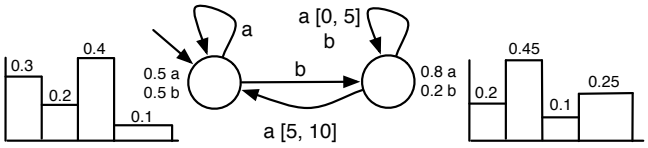

Fig. 2: A probabilistic DRTA. Every state is associated with a probability distribution over events and over time. The distribution over time is modeled using histograms. The bin sizes of the histograms are predetermined but left out for clarity.

In addition to choosing how to model the time and symbol distributions, we need to decide whether to make these two distributions dependent or independent. It is common practice to make these distributions independent, see, e.g., [6]. In this case, the time distribution represents a distribution over the waiting (or sojourn) time of every state. In some cases, however, it makes sense to let the time spent in a state depend on the generated symbol. By modeling this dependence, the model can deal with cases where some events are generated more quickly than others. Unfortunately, this dependence comes with a cost: the size of the model is increased by a polynomial factor (the product of the sizes of the distributions). Due to this blowup, we require a lot more data in order to identify a similar sized PDRTA. This is our main reason for modeling these two distributions independently.

This results in the following PDRTA model:

Definition 2. *(PDRTA) A* probabilistic DRTA *(PDRTA)* $\mathcal{A}$ *is a quadruple* $\langle \mathcal{A}', H, \mathcal{S}, \mathcal{T} \rangle$*, where* $\mathcal{A}' = \langle Q, \Sigma, \Delta, q_0 \rangle$ *is a DRTA without final states,* H *is a finite set of bins (time intervals)* $[v, v']$*,* $v, v' \in \mathbb{N}$*, known as the histogram,* $\mathcal{S}$ *is a finite set of symbol probability distributions* $\mathcal{S}_q = \{Pr(S = a \mid q) \mid a \in \Sigma, q \in Q\}$*, and* $\mathcal{T}$ *is a finite set of time-bin probability distributions* $\mathcal{T}_q = \{Pr(T \in h \mid q) \mid h \in H, q \in Q\}$*.*

The DRTA without final states specifies the *structure* of the PDRTA. The symbol- and time-probabilities $\mathcal{S}$ and $\mathcal{T}$ specify the probabilistic properties of a PDRTA. The probabilities in these sets are called the *parameters* of $\mathcal{A}$. However, in every set $\mathcal{S}_q$ and $\mathcal{T}_q$, the value of one of these parameters follows from the others because their values have to sum to 1. Hence, there are $(|\mathcal{S}_q|-1)+(|\mathcal{T}_q|-1)$ parameters per state q of our PDRTA model.

The probability that the next time value equals t given that the current state is q is defined as

$$Pr(T = t \mid q) = \frac{Pr(T \in h \mid q)}{v' - v + 1}$$

where $h = [v, v'] \in H$ is such that $t \in [v, v']$. Thus, in every time-bin the probabilities of the individual time points are modeled uniformly. The probability of an observation (a, t) given that the current state is q is defined as

$$Pr(O = (a, t) \mid q) = Pr(S = a \mid q) \times Pr(T = t \mid q)$$

Thus, the distributions over events and time are modeled to be independent.[4]

The probability of the next state q' given the current state q is defined as

$$Pr(X = q' \mid q) = \sum_{\langle q,q',a,[v,v']\rangle \in \Delta} \sum_{t \in [v,v']} Pr(O = (a,t) \mid q)$$

Thus, the model is deterministic. A PDRTA models a distribution over timed strings $Pr(O^* = \tau)$, defined using the computation of a PDRTA:

Definition 3. *(PDRTA computation) A finite* computation *of a PDRTA* $A = \langle\langle Q, \Sigma, \Delta, q_0\rangle, H, \mathcal{S}, \mathcal{T}\rangle$ *over a timed string* $\tau = (a_1,t_1)\dots(a_n,t_n)$ *is a finite sequence*

$$q_0 \xrightarrow{(a_1,t_1)} q_1 \dots q_{n-1} \xrightarrow{a_n,t_n} q_n$$

such that for all $1 \leq i \leq n$, $\langle q_{i-1}, q_i, a_i, [n_i, n_i']\rangle \in \Delta$, *and* $t_i \in [n_i, n_i']$. *The* probability *of* τ *given* $\mathcal{A}$ *is defined as* $Pr(O^* = \tau \mid \mathcal{A}) = \prod_{1 \leq i \leq n} Pr(O = (a_i,t_i) \mid q_{i-1}, H, \mathcal{S}, \mathcal{T})$.

Example 2. Figure 2 shows a PDRTA $\mathcal{A}$. Let $H = \{[0,2];[3,4];[5,6];[7,10]\}$ be the histogram. In every bin the distribution over time values is uniform. We can use $\mathcal{A}$ as a predictor of timed events. For example, the probability of $(a,3)(b,1)(a,9)(b,5)$ is $Pr((a,3)(b,1)(a,9)(b,5)) = 0.5 \times \frac{0.2}{2} \times 0.5 \times \frac{0.3}{3} \times 0.8 \times \frac{0.25}{4} \times 0.5 \times \frac{0.4}{2} = 1.25 \times 10^{-5}$.

A PDRTA essentially models a certain type of distribution over timed strings. An input sample S_+ can be seen as a sample drawn from such a distribution. The problem of identifying a PDRTA then consists of finding the distribution that generated this sample. We now describe how we adapt RTI in order to identify a PDRTA from such a sample.

3 Identifying PDRTAs from positive data

In this section, we adapt the RTI algorithm for the identification of DRTAs from labeled data (see [11]) to the setting of positive data. The result is the RTI+ algorithm, which stand for *real-time identification from positive data*. Given a set of observed timed strings S_+, the goal of RTI+ is to find a PDRTA that describes the real-time process that generated S_+. Note that, because RTI+ uses statistics (occurrence counts) to find this PDRTA, S_+ is a multi-set, i.e., S_+ can contain the same timed string multiple times.

Like RTI (see [11] for details), RTI+ starts with an augmented prefix tree acceptor (APTA). However, since we only have positive data available, the APTA will not contain rejecting states. Moreover, since the points in time where the observations are stopped are arbitrary, it also does not contain accepting states. Thus, the initial PDRTA simply is the *prefix tree* of S_+, see Figure 3.

[4] Modeling dependencies between events and time values is possible but this comes with a cost: the number of parameters of the model is increased by a polynomial factor. This blowup also increases the amount of data required for identification.

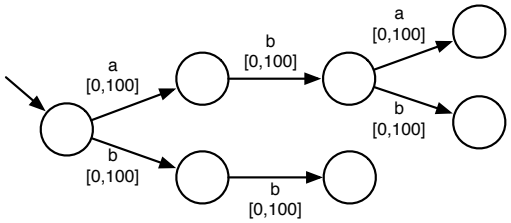

Fig. 3: A prefix tree. It is identical to an augmented prefix tree acceptor, but without accepting and rejecting states. The bounds of the delay guards are initialized to the minimum and maximum observed time value.

Starting from a prefix tree, our original algorithm tries to merge states and split transitions using a red-blue framework. A *merge* is the standard state-merging operation used in DFA identification algorithms such as ESDM [9]. A split can be seen as the opposite of a merge. A *split* of a transition δ requires a time value t and uses this to divide δ, its delay guard $[n, n']$, and the part of the PDRTA reached afterwards into two parts. The first part is reached by the timed strings that fire δ with a delay value less or equal to t, creating a new delay guard $[n, t]$. The second part is reached by timed strings for which this value is greater than t, creating delay guard $[t+1, n']$. The parts of the PDRTA reached after firing δ are reconstructed as new prefix trees, using the suffixes of the timed strings that reach these parts as input sample. See [11] for more information on the split operation.

RTI+ uses exactly the same operations and framework as RTI. The only difference is the evidence value we use. Originally, the evidence was based on the number of positive and negative examples that end in the same state. For RTI+, we require an evidence value that uses only positive examples, and that disregards which states these examples end in. We use a likelihood-ratio test for this purpose. We now describe this test and explain how we use it both as an evidence value and as a consistency check.

3.1 A likelihood-ratio test for state-merging

The likelihood-ratio test (see, e.g., [7]) is a common way to test nested hypotheses. A hypothesis H is called *nested* within another hypothesis H' if the possible distributions under H form a strict subset of the possible distributions under H'. Less formally, this means that H can be created by constraining H'. Thus, by definition H' has more unconstrained parameters (or degrees of freedom) than H. Given two hypotheses H and H' such that H is nested in H', and a data set S_+, the likelihood-ratio test statistic is computed by

$$\mathsf{LR} = \frac{\mathsf{likelihood}(S_+, H)}{\mathsf{likelihood}(S_+, H')}$$

where likelihood is a function that returns the *maximized likelihood* of a data set under a hypothesis, i.e., $\mathsf{likelihood}(S_+, H)$ is the maximum probability (with

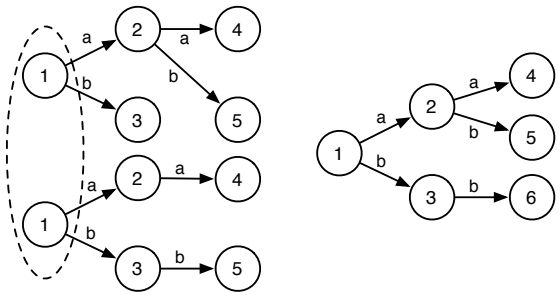

Fig. 4: The likelihood-ratio test. We test whether using the left model (two prefix trees) instead of the right model (a single prefix tree) results in a significant increase in the likelihood of the data with respect to the number of additional parameters (used to model the state distributions).

optimized parameter settings) of observing S_+ under the assumption that H was used to generate the data.

Let H and H' have n and n' parameters respectively. Since H is nested in H', the maximized likelihood of S_+ under H' is always greater than the maximized likelihood under H. Hence, the likelihood-ratio LR is a value between 0 and 1. When the difference between n and n' grows, the likelihood under H' can be optimized more and hence LR will be closer to 0. Thus, we can increase the likelihood of the data S_+ by using a different model (hypothesis) H', but at the cost of using more parameters $n'-n$. The likelihood-ratio test can be used to test whether this increase in likelihood is statistically significant. The test compares the value $-2\ln(\mathsf{LR})$ to a χ^2 distribution with $n'-n$ degrees of freedom. The result of this comparison is a p-value. A high p-value indicates that H is a better model since the probability that $n'-n$ extra parameters results in the observed increase in likelihood is high. A low p-value indicates that H' is a better model.

Applying the likelihood-ratio test to state-merging and transition-splitting is remarkably straightforward. Suppose that we want to test whether we should perform a merge of two states. Thus, we have to make a choice between two PDRTAs (models): the PDRTA $\mathcal{A}$ resulting from the merge of these states, and the PDRTA $\mathcal{A}'$ before merging these states. Clearly, $\mathcal{A}$ is nested in $\mathcal{A}'$. Thus all we need to do is compute the maximized likelihood of S_+ under $\mathcal{A}$ and $\mathcal{A}'$, and apply the likelihood-ratio test. Since PDRTAs are deterministic, the maximized likelihood can be computed simply by setting all the probabilities in the PDRTAs to their normalized counts of occurrence in S_+. We now show how to use this test in order to determine whether to perform a merge using an example.

Example 3. For simplicity, we disregard the time values of timed strings and the timed properties of PDRTAs. Suppose we want to test whether to merge the two root states of the prefix trees of Figure 4. These two prefix trees are parts of the PDRTA we are currently trying to identify. Hence only some strings from S_+ reach the top tree, and some reach the bottom tree.

Let $S = \{10 \times a, 10 \times aa, 20 \times ab, 10 \times b\}$ and $S' = \{20 \times aa, 20 \times bb\}$ be the suffixes of these strings starting from the point where they reach the root state of the top and bottom tree respectively, where $n \times \tau$ means that the (timed) string τ occurs n times. We first set all the parameters of the top tree in such a way that the likelihood of S is maximized: $p_{a,q_0} = \frac{4}{5}, p_{b,q_0} = \frac{1}{5}, p_{a,q_1} = \frac{1}{3}, p_{b,q_1} = \frac{2}{3}$ (this is easy because the model is deterministic). We do the same for the bottom tree and S': $p'_{a,q_0} = \frac{1}{2}, p'_{b,q_0} = \frac{1}{2}, p'_{a,q_1} = 1, p_{b,q_2} = 1$.

We can now compute the probability of S under the top tree: $p_1 = \left(\frac{4}{5}\right)^{40} \times \left(\frac{1}{5}\right)^{10} \times \left(\frac{1}{3}\right)^{10} \times \left(\frac{2}{3}\right)^{20} \approx 6.932 \times 10^{-20}$, and the probability of S' under the bottom tree $p_2 = \left(\frac{1}{2}\right)^{20} \times \left(\frac{1}{2}\right)^{20} \approx 9.095 \times 10^{-13}$. Next, we set the parameter of the right tree to maximize the likelihood of $S \cup S'$: $p_{a,q_0} = \frac{2}{3}, p_{b,q_0} = \frac{1}{3}, p_{a,q_1} = \frac{3}{5}, p_{b,q_1} = \frac{2}{5}, p_{b,q_2} = 1$, and compute the likelihood of the data under the right (merged) tree: $p_3 = \left(\frac{2}{3}\right)^{60} \times \left(\frac{1}{3}\right)^{30} \times \left(\frac{3}{5}\right)^{30} \times \left(\frac{2}{5}\right)^{20} \approx 3.211 \times 10^{-40}$. We multiply the top and bottom tree probabilities in order to get the likelihood of the data under the left (un-merged) tree, and use this to compute the likelihood-ratio: $LR = \frac{p_3}{p_1 \times p_2} \approx 5.093 \times 10^{-9}$.

The χ^2 value that we need to compare to a χ^2 distribution then becomes $\chi^2 \approx 38.19$. Per state $|\Sigma| - 1$ parameters are used. In the un-merged model, the number of (untimed) parameters is 5, in the merged model it is 3. A likelihood-ratio test using these values results in a p-value of 5.093×10^{-9}. This is a lot less than 0.05, and hence the merge results in a significantly worse model.

Testing whether to perform a split of a transition can be done in a similar way. When we want to decide whether to perform a split, we also have to make a choice between two PDRTAs: the PDRTA before splitting $\mathcal{A}$, and the PDRTA after splitting $\mathcal{A}'$. $\mathcal{A}$ is again nested in $\mathcal{A}'$, and hence we can perform the likelihood-ratio test in the same way.

3.2 Dealing with small frequencies

The likelihood-ratio test does not perform well when the tested models contain many unused parameters. The test tests whether an increase in the number of parameters leads to a significantly higher likelihood. Thus, if there are many unused parameters, this increase will usually not be significant. Hence, there will be a tendency to accept null-hypotheses, i.e., to merge states. This causes problems especially in the leafs of the prefix tree.

We deal with the issue of small frequencies by *pooling* the bins of the histogram and symbol distributions if the frequency of these bins in both states is less than 10. Pooling is the process of combining the frequencies of two bins into a single bin. In other words, we treat two bins as though it were a single one. For example, suppose we have three bins, and their frequencies are 7, 14, and 5, respectively. Then we treat it as being two bins with frequencies 12 and 14. In the likelihood-ratio test, this effectively reduces the amount of parameters of the tested models. Theoretically, it can be objected that this changes the model using the data. However, if we do not pool data, we will obtain too many

parameters for the states in which some bin occurrences are very unlikely. For instance, suppose we have a state in which 1000 symbols could occur, but only 10 of them actually occur. Then according to theory, we should count this state as having 999 parameters. We count it as having only 9.

3.3 The algorithm

We have just described the test we use to determine whether two states are similar. The null-hypothesis of this test is that two states are the same. When we obtain a p-value less than 0.05, we can reject this hypothesis with 95% certainty. When we obtain a p-value greater than 0.05, we cannot reject the possibility that the two states are the same. Instead of testing whether two states are the same, however, we want to test whether to perform a merge or a split, and if so, which one. When we test a merge, a high p-value indicates that the merge is good. When we test a split, a low p-value indicates that the split is good. We implemented this statistical evidence in RTI+ in a very straightforward way:

- If there is a split that results in a p-value less than 0.05, perform the split with the lowest p-value.
- If there is a merge that results in a p-value greater than 0.05, perform the merge with the highest p-value.
- Otherwise, perform a color operation.

Thus, we merge two states unless we are very certain that the two states are different. In addition, we always perform the merge or split that leads to the most certain conclusions.

In every iteration, RTI+ selects the most visited transition from a red state to a blue state and determines whether to merge the blue state, split the transition, or color the blue state red. The main reason for trying out only the most visited transition is that it reduces the run-time of the algorithm. Trying every possible merge and split would take much longer. Additionally, the tests performed using the most visited transition will be based on the largest amount of data. Hence, we are more confident that these conclusions are correct. An overview of the RTI+ algorithm is shown in Algorithm 1.

We claim that RTI+ is efficient, i.e., it that runs in polynomial time:

Proposition 1. *RTI+ is a polynomial-time algorithm.*

Proof. This follows from the fact that ID_1DTA is efficient [12] and the fact that every statistic can be computed (up to sufficient accuracy) in polynomial time for every state. Since, at any time during a run of the algorithm, the number of states does not exceed the size of the input, the proposition follows.

In addition to being time-efficient, we believe that RTI+ is also data-efficient. More specifically, we conjecture that returns a PDRTA that is equal to the correct PDRTA $\mathcal{A}_t$ in the limit. With equal we mean that these PDRTAs model the exact same probability distributions over timed strings.

Algorithm 1 Real-time identification from positive data: RTI+

Require: A multi-set of timed strings S_+ generated by a PDRTA $\mathcal{A}_t$
Ensure: The result is a small DRTA $\mathcal{A}$, in the limit $\mathcal{A} = \mathcal{A}_t$
  Construct a timed prefix $\mathcal{A}$ tree from S_+, color the start state q_0 of $\mathcal{A}$ red
  while $\mathcal{A}$ contains non-red states **do**
    Color blue all non-red target states of transitions with red source states
    Let $\delta = \langle q_r, q_b, a, g \rangle$ be most visited transition from a red to a blue state
    Evaluate all possible merges of q_b with red states
    Evaluate all possible splits of δ
    If the lowest p-value of a split is less than 0.05 **then** preform this split
    Else if the highest merge p-value is greater than 0.05 **then** perform this merge
    Else color q_b red
  end while

Conjecture 1. The result $\mathcal{A}$ of RTI+ converges efficiently in the limit to the correct PDRTA $\mathcal{A}'$ with probability 1.

Completeness of the algorithm follows from the fact that the algorithm is a special case of the ID_1DTA algorithm from [12]. The conjecture therefore holds if all correct merges and splits are performed given a input sample of size polynomial in the size of $\mathcal{A}'$.

The main reason for our conjecture follows from the fact that with increasing amounts of data, the p-value resulting from the likelihood-ratio test converges to 0 if the two states are different. Thus in the limit, RTI+ will perform all the necessary splits, and perhaps some more, and it will never perform an incorrect merge. However, when the two states tested in the likelihood-ratio test are the same, there is always a probability of 0.05 that the p-value is less than 0.05. Thus, at times it will not perform a merge when it should. Fortunately, not performing a merge or performing an extra split does not influence the language of the DRTA, or the distribution of the PDRTA. It only adds additional (unnecessary) states to the resulting PDRTA $\mathcal{A}$. Thus, in the limit, the algorithm should return a PDRTA $\mathcal{A}$ that is language equivalent to the target PDRTA $\mathcal{A}'$. Unfortunately, since we use multiple statistical tests that can become dependent, proving this conjecture is complex and left as future work.

4 Tests on artificial data

In order to evaluate the RTI+ algorithm, we test it on artificially generated data. First we generate a random PDRTA (without final states), and then we generate data using the distributions of this PDRTA. Unfortunately, it is difficult to measure the quality of models that are identified from such data. Commonly used measures include the predictive quality or a model selection criterion. However, such measures are meaningless on their own, they only useful to compare the performance of different methods against each other. Since, we know of no any other method for identifying a PDRTA, we cannot make use of these measures.

Therefore, in order to provide some insight into the capabilities of RTI+, we only show a typical result of RTI+ when run on this data.

We generate a random PDRTA with 8 states and a size 4 alphabet. Of the transitions of the PDRTA, 4 are split and assigned different target states at random. The number of possible time values for the timed strings is fixed at 100. The number of histogram bins used in the PDRTA is set to 10. Thus, there are individual probabilities for $[0, 9]$, $[10, 19]$, etc. The probabilities of these bins and the symbol bins are generated by first assigning to each bin a value between 0 and 1, drawn from a uniform distribution. These values are then normalized such that both the histogram values and the symbol values summed to 1. We generated 2000 timed strings from this PDRTA, which all have an exponentially distributed length with an average of 10.

Figure 5 shows the resulting original and identified PDRTA (no probability distributions are drawn). From this figure, it is clear that the most common mistake is the incorrect identification (or absence) of a clock guard. These are usually only minor errors, involving only infrequently visited transitions. The resulting PDRTA is thus very similar to the original used to generate the data. We performed such a test multiple times and using differently sized random PDRTAs. The results of these tests are encouraging for up to 8 states, a size 4 alphabet, and 4 splits. When either of these values is increased, the algorithm needs more than 2000 examples to come up with a similar PDRTA. These results are encouraging because PDRTAs of this size are complex enough to model interesting real-time systems.

5 Future work

In previous work, we described the RTI algorithm for identifying deterministic real-time automata (DRTAs) from labeled data. In this paper, we showed how to adapt it to the setting of positive data. The result is the RTI+ algorithm. RTI+ runs in polynomial time, and we conjecture that it converges efficiently to the correct probabilistic DRTA (PDRTA). In future work, we would like to prove this conjecture. This should be possible, because none of the statistics we use requires a large amount of data. Moreover, the fact that there exist polynomial characteristics sets for DRTAs (see [12]) should somehow extend to identifying PDRTAs.

RTI+ uses a likelihood-ratio test in order to determine which states to merge and which transitions to split. Although this test is designed for the purpose of identifying a PDRTA from positive data, it can easily be modified in order to identify probabilistic DFAs. It would be interesting to test such an approach.

The achieved performance of RTI+ is shown to be sufficient in order to identify complex real-time systems. We believe this performance to be sufficient to be useful for identifying real-world real-time systems. We invite everyone with timed data to try RTI+ to identify behavioral models, and network protocols. The source code of RTI+ is available on-line from the first author's homepage.

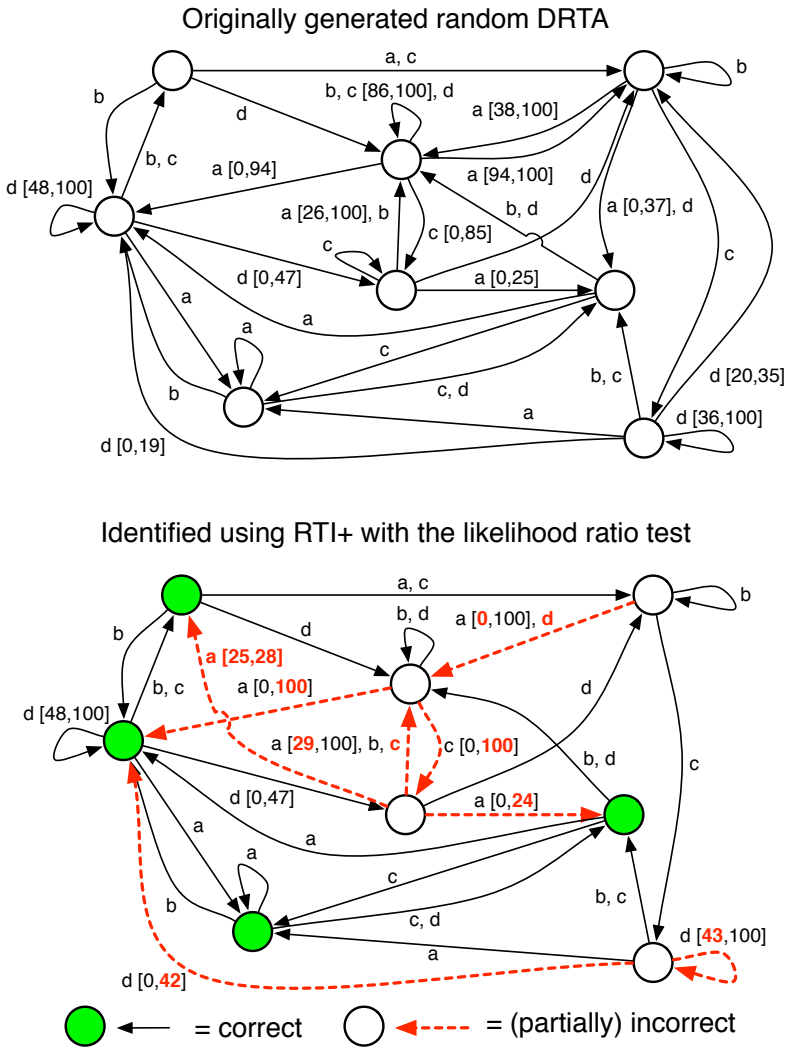

Fig. 5: A randomly generated DRTA (top) and the DRTA identified by our algorithm (bottom). The dashed lines are (partially) incorrectly identified transitions. The solid states are correctly identified, including all outgoing transitions.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *ICGI*, volume 862 of *LNCS*, pages 139–150. Springer, 1994.
3. A. Clark and F. Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, pages 473–497, 2004.
4. C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.
5. C. Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6(1):2–23, 2001.
6. Y. Guédon. Estimating hidden semi-Markov chains from discrete sequences. *Journal of Computational and Graphical Statistics*, 12(3):604–639, 2003.
7. W. L. Hays. *Statistics*. Wadsworth Pub Co, fifth edition, 1994.
8. C. Kermorvant and P. Dupont. Stochastic grammatical inference with multinomial tests. In *ICGI*, volume 2484 of *LNAI*, pages 149–160. Springer, 2002.
9. K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *ICGI*, volume 1433 of *LNCS*. Springer, 1998.
10. M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
11. S. Verwer, M. de Weerdt, and C. Witteveen. An algorithm for learning real-time automata. In *Benelearn*, pages 128–135, 2007.
12. S. Verwer, M. de Weerdt, and Cees Witteveen. One-clock deterministic timed automata are efficiently identifiable in the limit. In *LATA*, volume 5457 of *LNCS*, pages 740–751. Springer, 2009.