Inference of Event-Recording Automata using Timed Decision Trees

Olga Grinchtein
IT Department
Uppsala University
Uppsala, Sweden
olgag@it.uu.se

Bengt Jonsson
IT Department
Uppsala University
Uppsala, Sweden
bengt@it.uu.se

Abstract

In *regular inference*, the problem is to infer a regular language, typically represented by a deterministic finite automaton (DFA) from answers to a finite set of membership queries, each of which asks whether the language contains a certain word. There are many algorithms for learning DFAs, the most well-known being the L^* algorithm due to Dana Angluin. However, there are almost no extensions of these algorithms to the setting of timed systems. We present an algorithm for inferring a model of a timed system using Angluin's setup. One of the most popular model for timed system is timed automata. Since timed automata can freely use an arbitrary number of clocks, we restrict our attention to systems that can be described by *event-recording automata* (DERAs). In previous work, we have presented an algorithm for inferring a DERA in the form of a region graph. In this paper, we present a novel inference algorithm for DERAs, which avoids constructing a (usually prohibitively large) region graph. We must then develop techniques for inferring guards on transitions of a DERA. Our construction deviates from previous work on inference of DERAs in that it first constructs a so called timed decision tree from observations of system behavior, which is thereafter folded into an automaton.

1 Introduction

Research during the last decades has developed powerful techniques for using models of reactive systems in specification, automated verification (e.g., [CGP99]), test case generation (e.g., [FJJV97, SEG00]), implementation (e.g., [HLN+90]), and validation of reactive systems in telecommunication, embedded control, and related application areas. Typically, such models are assumed to be developed *a priori* during the specification and design phases of system development. In practice, however, often no formal specification is available, or becomes outdated as the system evolves over time. One must then construct a model that describes the behavior of an existing system or implementation. In software verification, techniques are being developed for generating abstract models of software

modules by static analysis of source code (e.g., [CDH+00, Hol00]). However, peripheral hardware components, library modules, or third-party software systems do not allow static analysis. In practice, such systems must be analyzed by observing their external behavior. Techniques for constructing finite-state models by analysis of externally observable behavior (black-box techniques) have been used, e.g., in regression testing [HHNS02, HNS03]), and in model checking [GPY02] of finite-state systems for which no model or source code is available.

The construction of models from observations of system behavior can be seen as a learning problem. For finite-state reactive systems, this can be formulated as the problem of *regular inference:* to infer a (deterministic) finite automaton by posing a finite set of *membership queries*, each of which asks whether a certain word is accepted by the automaton or not. There are several slightly different algorithms for regular inference (e.g., [Ang87, Gol67, KV94, RS93, BDG97]), which guarantee that a correct automaton will be constructed if "sufficiently many" membership queries have been posed. In some settings, such as in the L^* algorithm [Ang87], the inference algorithm may also pose *equivalence queries* that ask whether a hypothesized automaton is equivalent to the one that is being investigated: such a query is answered either by *yes* or by a counterexample on which the hypothesis and the correct automata disagree.

In this paper, we extend the inference algorithm of Angluin and others to the setting of timed systems, more precisely systems that can be described by a timed automaton [AD94], i.e., a finite automaton equipped with clocks that constrain the times of occurrences of actions. One motivation is to develop techniques for creating abstract timed models of hardware components, device drivers, etc., which can then be used in analysis of timed reactive systems. Timed automata have several characteristics that are problematic for the design of inference algorithms: it is not observable how many clocks they use, nor when the clocks are reset, and timed automata can not in general be determinized [AD94]. We therefore restrict consideration to the class of *event-recording automata* [AFH99]. These are timed automata that, for every action a, use a clock that records the time of the last occurrence of a. Event-recording automata can be determinized, and are sufficiently expressive to model many interesting timed systems; for instance, they are as powerful as timed transition systems [HMP94, AFH99], another popular model for timed systems. We assume that an inference algorithm observes a system by checking whether certain actions can be performed at certain moments in time, and that it is able to control and record precisely the timing of the occurrence of each action.

In our earlier work [GJL05], we have designed an inference technique for event-recording automata, which infers explicit representation of the region graph [AD94]. The region graph can be viewed as an ordinary automaton, and be inferred using techniques for untimed regular inference, but at the cost of a high blow-up: for instance, each transition will be enabled in only one clock region causing an explosion in the number of generated transitions. In this paper, we avoid the region graph construction; one goal is that guards on transitions should not depend on clocks that are not relevant for the occurrence

of current or future actions. We must then develop techniques for inferring "relevant" guards on transitions of a timed automaton.

A novel feature of our algorithm (in comparison with other regular inference algorithms) is that it contains two constructions: the first construction generates a so-called *timed decision tree* from the answers to the membership queries that have been performed so far. The conditions (or guards) that distinguish between branches in this tree should correspond to guards in the inferred automaton. A guard is therefore introduced only if there is a pair of observations with different outcomes, where this guard is the only means to distinguish between them. The algorithm includes a systematic search for such pairs of observations. The second construction consists in folding the timed decision tree into an event-recording automaton by appropriate merging of nodes, in an analogous manner as in algorithms for (untimed) regular inference.

Related Work In previous work, authors of this paper [GJL04] have presented algorithms for the restricted class of event-recording automata that have at most one outgoing transition per action; under this restriction, the time of an action occurrence depends only on the (untimed) past sequence of actions, and not on their timing. Later [GJL05] the same authors presented an algorithm for inferring of the full class of event-recording automata. This is much more challenging, since it must be inferred how the timing of an action occurrence may depend on timing of previous actions. A drawback of the algorithm is that it constructs a region graph, thus causing an explosion in the number of states and transitions.

Verwer et. al [VdWW06] present an algorithm for passive learning of timed automata with one clock which is reset at every transition. In passive learning an automaton is constructed from a given set of observations and no queries are performed. Passive learning even for this class of timed automata is a hard problem, since one must decide how to introduce guards in the automaton. The algorithm constructs a prefix tree from a timed sample and then tries to merge nodes of this tree pairwise to form an automaton. If the resulting automaton does not agree with the sample then the last merge is undone and a new merge is attempted. The algorithm does not construct timed automata in a systematic way, and it is hard to generalize the algorithm to timed automata with more than one clock.

A related field to timed automata learning is conformance testing for real-time systems. It was shown in [BGJ$^+$05] that if a set of examples form a conformance test suite for a finite state machine M, then M can be inferred from these set of examples using automata learning techniques. Conversely, it was also shown that if a finite state machine M is inferred from a set of examples, and furthermore M is the only such automaton, then the set of examples forms a conformance test suite for M. Springintveld et al. [SVD01] introduce an algorithm which generates a conformance test suite for timed automata. The algorithm constructs a so-called grid automaton, which is obtained by discretizing the time domain with the help of time transitions, each of which advances

time by 2^{-n} time units for some sufficiently large natural number n. Then the Vasilevskii-Chow algorithm [Cho78], [Vas73] is applied to the grid automaton to generate a conformance test suite.

Several papers are concerned with finding a definition of timed languages which is suitable as a basis for inference algorithms. There are several works that define determinizable classes of timed automata (e.g., [AFH99, SV96]) and right-congruences of timed languages (e.g., [MP04, HRS98, Wil94]), motivated by testing and verification.

Structure of Paper The paper is structured as follows. After preliminaries and a definition of event-recording automata in the next section, the construction and maintenance of timed decision trees is described in Sections 4 and 5, followed by a technique for folding them into automata in Section 6. The overall algorithm is summarized in Section 7, accompanied by an analysis of the query complexity of our algorithm. An illustrating example is given in Section 8, and finally Section 9 concludes the paper.

2 Preliminaries

We write $\mathbb{R}^{\geq 0}$ for the set of nonnegative real numbers, and $\mathbb{N}$ for the set of natural numbers. Let Σ be a finite alphabet of size $|\Sigma|$. A *timed word* over Σ is a finite sequence $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ of symbols $a_i \in \Sigma$ paired with nonnegative real numbers $t_i \in \mathbb{R}^{\geq 0}$ such that the sequence $t_1 t_2 \dots t_n$ of time-stamps is nondecreasing. We use λ to denote the empty word and $|w_t|$ to denote the number of pairs in w_t.

An event-recording automaton contains for every symbol $a \in \Sigma$ a clock x_a, called the *event-recording clock* of a. Intuitively, x_a records the time elapsed since the last occurrence of the symbol a. We write C_Σ for the set $\{x_a \mid a \in \Sigma\}$ of event-recording clocks. A *clock valuation* γ is a mapping from C_Σ to $\mathbb{R}^{\geq 0}$. We define $\gamma[x_a \to 0]$ to be the clock valuation γ' such that $\gamma'(x_a) = 0$ and $\gamma'(x_b) = \gamma(x_b)$ for all $b \neq a$.

Throughout the paper, we will use an alternative, equivalent representation of timed words, namely clocked words. A *clocked suffix word* over Σ is a sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ that are paired with clock valuations, which for all $a \in \Sigma$ satisfies

$$\gamma_i(x_a) = \gamma_{i-1}(x_a) + \gamma_i(x_{a_{i-1}}) \text{ whenever } 1 < i \leq n \text{ and } a \neq a_{i-1}. \tag{1}$$

A *clocked word* over Σ is a clocked suffix word $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ which in addition satisfies

$$\gamma_1(x_a) = \gamma_1(x_b) \text{ for all } a, b \in \Sigma. \tag{2}$$

Each timed word $(a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ can be naturally transformed into the clocked word $(a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ where for each i with $1 \leq i \leq n$,

- $\gamma_i(x_a) = t_i$ if $a_j \neq a$ for $1 \leq j < i$,

- $\gamma_i(x_a) = t_i - t_j$ if there is a j with $1 \leq j < i$ and $a_j = a$, such that $a_k \neq a$ for $j < k < i$ (i.e. time elapsed since last occurrence of the symbol a in the timed word).

Conversely, each clocked word can also be transformed into the timed word. A *timed language* over Σ is a set of timed words (or equivalently clocked words) over Σ.

A *clock constraint* is a conjunction of atomic constraints of the form $x \sim k$ or $x - y \sim k$ for $x, y \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$, and $k \in \mathbb{N}$. A clock constraint is *K-bounded* if it contains no constant larger than K. We use $\gamma \models \varphi$ to denote that the clock valuation γ satisfies the clock constraint φ, defined in the usual manner. A *clock guard* is a clock constraint whose conjuncts are only of the form $x \sim k$ (for $x \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$), i.e., comparison between clocks is not permitted. The set of clock guards is denoted by G_Σ. A K-bounded *simple clock guard* is a clock constraint whose conjuncts are only of the form $x = k$, $k' < x < k' + 1$ or $x > K$ for $x \in C_\Sigma$, $0 \leq k \leq K$ and $0 \leq k' \leq K - 1$.

A *guarded word* is a sequence $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ of symbols $a_i \in \Sigma$ that are paired with clock guards. For a clocked suffix word $w_c = (a_1, \gamma_1)(a_2, \gamma_2) \ldots (a_n, \gamma_n)$ we use $w_c \models w_g$ to denote that $\gamma_i \models g_i$ for $1 \leq i \leq n$. We say that w_c *leads to* w_g or that w_c *is in* w_g if $w_c \vDash w_g$. We say that w_c *passes* w_g if $w'_c \vDash w_g$ for some prefix w'_c of w_c. If for all i, g_i is a K-bounded simple clock guard, w_g is called *K-bounded simple guarded word*. We omit K-bounded if K is clear from the context.

For a guarded word w_g, we introduce the *strongest postcondition* of w_g, denoted by $sp(w_g)$, as the constraint on clock values that are induced by w_g on any following occurrence of a symbol. Postcondition computation is central in techniques for symbolic verification of timed automata [BDM+98, BLL+96], and can be done inductively as follows:

- $sp(\lambda) = \bigwedge_{a,b \in \Sigma} x_a = x_b$,
- $sp(w_g(a, g)) = ((sp(w_g) \wedge g)[x_a \mapsto 0]) \uparrow$,

where for clock constraint φ and clock x,

- $\varphi[x \mapsto 0]$ is the condition $x = 0 \wedge \exists x.\varphi$,
- $\varphi \uparrow$ is the condition $\exists d.\varphi'$, where d ranges over $\mathbb{R}^{\geq 0}$ and where φ' is obtained from φ by replacing each clock y by $y - d$.

Both operations can be expressed as corresponding operations on clock constraints [DT98].

For each clock constraint φ there are in general several other clock constraints that are equivalent to φ in the sense that they identify the same set of clock valuations. If φ is satisfiable, there is among these a unique *canonical* clock constraint, denoted $Can(\varphi)$, obtained by closing φ under all consequences of pairs of conjuncts in φ, i.e.,

- from two difference bounds, such as $x_a - x_b \leq 2$ and $x_b - x_c < 3$, we derive a new difference bound, viz. $x_a - x_c < 5$, and

- from a difference bound and a clock bound, such as $x_a - x_b \leq 2$ and $x_a \geq 3$, we derive a new clock bound, viz. $x_b \geq 1$,
- from an upper and a lower clock bound, such as $x_a \leq 3$ and $x_b > 2$, we derive a new difference bound, viz. $x_a - x_b < 1$,

until saturation, and thereafter keeping the tightest bounds for each clock and each clock difference. If the canonical form contains inconsistent constraints, or requires some clock to be negative, then the clock constraint is unsatisfiable. We define the canonical form for an unsatisfiable clock constraint to be *false* [Dil89].

For $t \in \mathbb{R}^{\geq 0}$, $fract(t)$ denotes the fractional part of t, $\lfloor t \rfloor$ denotes the integer part of t, and $\lceil t \rceil = \lfloor t \rfloor + 1$. Given a positive non-negative K, we define the *region equivalence*, denoted $\sim_K$, on the set of clock valuations by $\gamma \sim_K \gamma'$ if

- for all $x_a \in C_\Sigma$, either
  - $\gamma(x_a)$ and $\gamma'(x_a)$ are both greater than K, or
  - $\lfloor \gamma(x_a) \rfloor = \lfloor \gamma'(x_a) \rfloor$ and $fract(\gamma(x_a)) = 0$ iff $fract(\gamma'(x_a)) = 0$,

  and
- for all $x_a, x_b \in C_\Sigma$ with $\gamma(x_a) \leq K$ and $\gamma(x_b) \leq K$, $fract(\gamma(x_a)) \leq fract(\gamma(x_b))$ iff $fract(\gamma'(x_a)) \leq fract(\gamma'(x_b))$.

A *region* is an equivalence class of clock valuations induced by $\sim_K$. We denote by $[\gamma]_K$ the region of γ. The number of regions R_K is bounded by $|\Sigma|!2^{|\Sigma|}(2K+2)^{|\Sigma|}$ [AD94]. A region $[\gamma]_K$ is *initial* if $\gamma(x_a) = 0$ for some $a \in \Sigma$.

Region equivalence induces a natural equivalence on clock constraints. We say that two clock constraints φ and φ' are *region-equivalent*, denoted $\varphi \approx_K \varphi'$, if for each clock valuation γ with $\gamma \vDash \varphi$ there is a clock valuation γ' with $\gamma' \vDash \varphi'$ such that $\gamma \sim_K \gamma'$, and vice versa.

Definition 2.1 *An* event-recording automaton *(ERA) over Σ is a tuple $\mathcal{A} = \langle L, L^0, L^f, E \rangle$ consisting of a finite set L of* locations*, a set $L^0 \subseteq L$ of* start locations*, a set L^f of accepting locations, and a finite set E of edges. Each edge is a quadruple (l, l', a, g) with a source location $l \in L$, a target location $l' \in L$, an input symbol $a \in \Sigma$, and a clock guard $g \in G_\Sigma$.* □

We write $l \overset{(a,g)}{\rightarrow} l'$ to denote $(l, l', a, g) \in E$.

For a guarded word $w_g = (a_1, g_1) \ \ldots \ (a_n, g_n)$, we use $l_0 \overset{w_g}{\Longrightarrow} l_n$ to denote that there is a sequence

$$l_0 \overset{(a_1,g_1)}{\rightarrow} l_1 \overset{(a_2,g_2)}{\rightarrow} l_2 \cdots \overset{(a_{n-1},g_{n-1})}{\rightarrow} l_{n-1} \overset{(a_n,g_n)}{\rightarrow} l_n$$

of locations $l_i \in L$ and edges $(l_{i-1}, l_i, a_i, g_i) \in E$. For a location $l \in L$, let $\mathcal{L}_g^+(\mathcal{A}, l)$ be the set of guarded words w_g such that $l \overset{w_g}{\Longrightarrow} l'$ for some accepting $l' \in L^f$, and let $\mathcal{L}_g^-(\mathcal{A}, l)$ be the set of guarded words w_g such that $l \overset{w_g}{\Longrightarrow} l'$ for

some nonaccepting $l' \notin L^f$. Let $\mathcal{L}_c^+(\mathcal{A}, l)$ be the set of clocked suffix words w_c such that $w_c \vDash w_g$ for some $w_g \in \mathcal{L}_g^+(\mathcal{A}, l)$. The clocked word w_c is *accepted* by $\mathcal{A}$ if $w_c \in \mathcal{L}_c^+(\mathcal{A}, l^0)$ where $l^0 \in L^0$, otherwise w_c is *rejected* by $\mathcal{A}$. The timed language $\mathcal{L}(\mathcal{A})$ defined by $\mathcal{A}$ consists of all clocked words accepted by $\mathcal{A}$. Two ERAs $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

An ERA is *time-deterministic* (TDERA) iff it has a single start location and for each location l and input symbol a, the guards $g_1, \ldots, g_n$ of edges from l labeled a are total and mutually exclusive, i.e., $g_1 \vee \ldots \vee g_n \equiv \mathit{true}$ and $g_i \wedge g_j \equiv \mathit{false}$ whenever $1 \leq i < j \leq n$.

3 Overview of Our Inference Algorithm

The algorithm presented in this paper is designed to infer a TDERA which is equivalent to a given TDERA $\mathcal{A}$. In this algorithm, which follows the setup of Angluin's algorithm [Ang87], a so called *Learner*, who initially knows Σ and the greatest constant K appearing in clock guards of $\mathcal{A}$, is trying to learn $\mathcal{L}(\mathcal{A})$ by asking queries to a *Teacher*, who knows $\mathcal{A}$. There are two kinds of queries:

- A *membership query* consists in asking whether a clocked word w_c over Σ is in $\mathcal{L}(\mathcal{A})$.
- An *equivalence query* consists in asking whether a hypothesized TDERA $\mathcal{H}$ is correct, i.e., whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{A})$. The *Teacher* will answer *yes* if $\mathcal{H}$ is correct, or else supply a *counterexample*, i.e, a clocked word w_c which is either in $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{H})$ or in $\mathcal{L}(\mathcal{H}) \setminus \mathcal{L}(\mathcal{A})$.

The typical behavior of a *Learner* is to start by asking a sequence of membership queries, and gradually build a hypothesized TDERA $\mathcal{H}$ using the obtained answers. When the *Learner* feels that she has built a hypothesis $\mathcal{H}$, she makes an equivalence query to find out whether $\mathcal{H}$ is correct. If the answer is *yes*, the *Learner* has succeeded, otherwise she uses the returned counterexample to revise $\mathcal{H}$ and perform subsequent membership queries until arriving at a new hypothesized TDERA, etc. The algorithm must guarantee that there is a bound, which depends on $\mathcal{A}$, on the number of performed membership and equivalence queries needed before the *Learner* is able to make a successful equivalence query.

Let us represent the information gained by the *Learner* at any point during the learning process as a partial mapping Obs from clocked words to $\{+, -\}$, where $+$ stands for *accepted* and $-$ for *rejected*. The domain $Dom(Obs)$ of Obs is the set of clocked words for which membership queries have been performed, or which have been supplied as counterexamples in response to equivalence queries. We will consider the case where $Dom(Obs)$ is initially empty, but our techniques work equally well when $Dom(Obs)$ is some given initial set of observations. An inference algorithm should prescribe how, starting from any given mapping Obs, additional membership queries are performed in order to gather sufficient information for constructing a "good hypothesis" in the form of a TDERA $\mathcal{H}$, which can be supplied in an equivalence query. A natural requirement on the

constructed TDERA $\mathcal{H}$ is that it *agrees* with Obs, in the sense that any clocked word $w_c \in Dom(Obs)$ is accepted by $\mathcal{H}$ if $Obs(w_c) = +$, and rejected by $\mathcal{H}$ if $Obs(w_c) = -$.

During the gathering of information by membership queries, our algorithm organizes the information in Obs in a so-called *timed decision tree* (defined in the next section), which roughly can be regarded as a prefix of the "unrolling" of the (later) constructed hypothesized TDERA $\mathcal{H}$ into an infinite tree. A main problem when constructing the timed decision tree is to find suitable guards; these should ideally be the guards that occur on the corresponding edges of the TDERA $\mathcal{A}$. However, the *Learner* does not know *a priori* which guards to use in the final TDERA. Therefore, guards will be introduced "by need", when the information in Obs shows that they are necessary. Initially, the edges in the tree will use only *true* in guards. Any non-trivial guard in the timed decision tree will be obtained from a pair of clocked words in $Dom(Obs)$, one of which is rejected and one of which is accepted, and which are so "close" that the obtained guard is the only means to separate them.

The goal of the information gathering through membership queries is that Obs and the timed decision tree should attain four desirable properties: *well-formedness* and *consistency*, defined in Section 4, and *suffix-completeness* and *region-consistency*, defined in Section 6. When these properties have been attained, the tree will be folded into a hypothesized TDERA by merging nodes which have "similar" future behavior according to Obs and the tree, as will be described in Section 6.

4 Timed Decision Trees.

In this section, we present timed decision trees, which are used to organize the results of queries.

Definition 4.1 *A* timed decision tree *is a prefix-closed finite set $\mathcal{N}$ of guarded words, such that for any guarded word $w_g \in \mathcal{N}$ and symbol a, the set of guarded words $w_g(a, g_1), \ldots, w_g(a, g_n)$ in $\mathcal{N}$ that extend w_g by the symbol a is either empty, or has the properties that $g_1 \vee \cdots \vee g_n \equiv true$ and $g_i \wedge g_j \equiv false$ whenever $i \neq j$.* □

A timed decision tree $\mathcal{N}$ is intended to be seen as a prefix of an unfolding of a TDERA. Each guarded word in $\mathcal{N}$ can then be viewed as the sequence of labels on a sequence of edges from the initial location of the TDERA to some location in its unfolding. We will often refer to the elements of a timed decision tree as *nodes*. The structure into which results of queries are organized is obtained by combining a timed decision tree with the results of performed queries.

Definition 4.2 *An* observation structure *is a pair $\langle Obs, \mathcal{N} \rangle$, where Obs is a partial mapping from clocked words to $\{+, -\}$ and $\mathcal{N}$ is a timed decision tree.* □

The mapping in an observation structure records whether individual clocked words are accepted or rejected, but we extend it to a mapping from nodes in a timed decision tree, as follows. For a partial mapping Obs from clocked words to $\{+, -\}$ and a guarded word w_g, define

$$Obs(w_g) = \begin{cases} \top & \text{if there are } w_c, w_c' \in Dom(Obs) \text{ with } w_c \vDash w_g \\ & \quad \text{and } w_c' \vDash w_g \text{ such that } Obs(w_c) \neq Obs(w_c') \\ Obs(w_c) & \text{if there is } w_c \in Dom(Obs) \text{ with } w_c \vDash w_g \text{ and} \\ & \quad Obs(w_c') = Obs(w_c) \text{ for all } w_c' \in Dom(Obs) \\ & \quad \text{with } w_c' \vDash w_g \\ \bot & \text{if there is no } w_c \in Dom(Obs) \text{ with } w_c \vDash w_g \end{cases}$$

By this definition, the function Obs is overloaded so that it can also be regarded as a labeling of nodes in a timed decision tree. A node is labeled by $+$ (or $-$) if every observation that leads to the node is accepted (or rejected). A node is labeled by $\top$, and called *inconsistent*, (w.r.t. Obs), if there are both accepted and rejected observations that lead to the node. We say that Obs *distinguishes* between w_c and w_c' if $Obs(w_c) \neq Obs(w_c')$. If there are no observations that lead to the node, the node is labeled by $\bot$.

Definition 4.3 *An observation structure $\langle Obs, \mathcal{N} \rangle$ is*

- well-formed *if*
  - *for any $w_c \in Dom(Obs)$ there is a $w_g \in \mathcal{N}$ such that $w_c \vDash w_g$, and*
  - *$Obs(w_g) \neq \bot$ for all $w_g \in \mathcal{N}$,*
- consistent *if $\mathcal{N}$ has no inconsistent nodes w.r.t. Obs.* □

In the following, we will assume that all observation structures are well-formed, since it is easy to make them well-formed either by adding new nodes that extend parents by suffixes of form $(a_1, true) \cdots (a_m, true)$, or by asking additional membership queries, e.g., for prefixes of clocked words in $Dom(Obs)$. Making an observation structure consistent is more complicated. It typically involves asking additional membership queries to find new guards, and restructuring of the tree, as described in Section 5.

Example 4.4 Figure 1 shows an example of a well-formed observation structure. To the left is the mapping Obs; to make the notation more compact we use timed words as a short representation of clocked words. To the right is the timed decision tree. We label the edges of the timed decision tree with pairs of the form (a, g) where a is a symbol and g is a clock guard. The guarded word that constitutes a node is the sequence of labels on the path reaching the node. Each node w_g in the tree is labeled by $Obs(w_g)$.

The only nontrivial guards occur on the two rightmost edges, which separate occurrences of the symbol b depending on whether $x_a = 0$ or $x_a > 0$. This guard is motivated by the observations for the two timed words $(a, 0.5)(b, 0.5)$ and

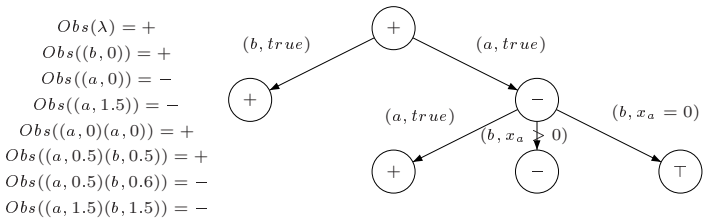


Figure 1: An observation structure $\langle Obs, \mathcal{N} \rangle$, with Obs to the left and $\mathcal{N}$ to the right.

.

$(a, 0.5)(b, 0.6)$ which are distinguished by Obs. The rightmost node $(a, true)(b, x_a = 0)$ is inconsistent since two timed words $(a, 0.5)(b, 0.5)$ and $(a, 1.5)(b, 1.5)$, which are distinguished by Obs, lead to it. □

In the next section, we describe how inconsistent nodes, such as the rightmost one in Figure 1, are removed by finding new guards and restructuring the tree after having performed additional membership queries.

5 Removing Inconsistent nodes

In this section, we describe our procedure for removing inconsistent nodes in a timed decision tree by finding new guards and restructuring the tree accordingly. New guards are inferred from so-called *critical pairs*, described in Section 5.1, which are pairs of "close" clocked words which are distinguished by Obs and uniquely suggest a guard that separates them, as described in Section 5.2. In Section 5.3 we present a procedure, inspired by binary search, for finding critical pairs whenever the tree contains an inconsistent node, and in Section 5.4 we describe how to restructure the tree to avoid the initial inconsistency.

5.1 Critical Pairs

Critical pairs are pairs of "close" clocked words, used to suggest guards in a timed decision tree. To define them, we first formalize the notion of "close" observations. For two nonnegative real numbers $p, p' \in \mathbb{R}^{\geq 0}$, define $p \approx_K p'$ if either

- $p > K$ and $p' > K$, or
- both p and p' are integers with $p = p'$, or
- both p and p' are non-integer with $\lfloor p \rfloor = \lfloor p' \rfloor$ (i.e., they have the same integer parts).

For two clock valuations γ and γ', define $\gamma \approx_K \gamma'$ if $\gamma(x_a) \approx_K \gamma'(x_a)$ for all $a \in \Sigma$. For two clocked (suffix) words $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ and $w'_c = (a'_1, \gamma'_1) \dots (a'_n, \gamma'_n)$, define $w_c \approx_K w'_c$ if $a_i = a'_i$ and $\gamma_i \approx_K \gamma'_i$ for all $1 \leq i \leq n$.

Definition 5.1 *A pair (w_c, w'_c) of clocked words, where $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ and $w'_c = (a_1, \gamma'_1) \dots (a_n, \gamma'_n)$ is* adjacent *if $w_c \not\approx_K w'_c$, and for all i with $1 \leq i \leq n$ and $a \in \Sigma$, either $\gamma_i(x_a) > K$ and $\gamma'_i(x_a) > K$, or*

- *whenever $\gamma'_i(x_a)$ is an integer, then $\gamma_i(x_a) = \gamma'_i(x_a)$,*
- *whenever $\gamma_i(x_a)$ is an integer, and $\gamma'_i(x_a)$ is not, then $|\gamma_i(x_a) - \gamma'_i(x_a)| < 1$,*
- *whenever $\gamma_i(x_a)$ is not an integer, then $\gamma_i(x_a) \approx_K \gamma'_i(x_a)$.* □

Intuitively, a pair (w_c, w'_c) of clocked words is adjacent if, for each position i and symbol a for which $\gamma_i(x_a) \not\approx_K \gamma'_i(x_a)$, there is exactly one atomic guard which can separate $\gamma_i(x_a)$ from $\gamma'_i(x_a)$. If there is only one position i and symbol a with $\gamma_i(x_a) \not\approx_K \gamma'_i(x_a)$, then this atomic guard is unique, but in general there can be several such positions and symbols. Note that adjacency is not a symmetric relation, i.e., if (w_c, w'_c) is adjacent, then (w'_c, w_c) is not adjacent.

As a measure of "closeness" between clocked words in adjacent pairs (w_c, w'_c), we define $\mathit{diff}(w_c, w'_c)$ to be the set of quadruples $(i, x_a, k, \geq)$ and $(i, x_a, k, \leq)$ with $k \leq K$, such that

- $(i, x_a, k, \geq) \in \mathit{diff}(w_c, w'_c)$ iff $\gamma_i(x_a) = k$ and $\gamma'_i(x_a) < k$ and
- $(i, x_a, k, \leq) \in \mathit{diff}(w_c, w'_c)$ iff $\gamma_i(x_a) = k$ and $\gamma'_i(x_a) > k$.

Example 5.2 As an illustration of Definition 5.1, (w_c, w'_c) is an adjacent pair when

$$w_c = (a, \left(\begin{smallmatrix} x_a \to 0.8 \\ x_b \to 0.8 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.2 \\ x_b \to 1.0 \end{smallmatrix} \right)) \text{ and}$$
$$w'_c = (a, \left(\begin{smallmatrix} x_a \to 0.8 \\ x_b \to 0.8 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.3 \\ x_b \to 1.1 \end{smallmatrix} \right)) \quad .$$

We have $\mathit{diff}(w_c, w'_c) = \{(2, x_b, 1, \leq)\}$. If Obs distinguishes w_c and w'_c, then it is clear that the tree must contain the conjuncts $x_b > 1$ and $x_b \leq 1$ in guards on two different outgoing edges from the node to which $(a, \left(\begin{smallmatrix} x_a \to 0.8 \\ x_b \to 0.8 \end{smallmatrix} \right))$ leads, in order to separate the observations. □

Example 5.3 A more complicated example of an adjacent pair is (w_c, w'_c) where

$$w_c = (a, \left(\begin{smallmatrix} x_a \to 1.0 \\ x_b \to 1.0 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.5 \\ x_b \to 1.5 \end{smallmatrix} \right)) \text{ and}$$
$$w'_c = (a, \left(\begin{smallmatrix} x_a \to 0.5 \\ x_b \to 0.5 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.6 \\ x_b \to 1.1 \end{smallmatrix} \right))$$

Here $\mathit{diff}(w_c, w'_c) = \{(1, x_a, 1, \geq), (1, x_b, 1, \geq)\}$. If Obs distinguishes w_c and w'_c, we can choose either the guard $x_a \geq 1$ or the guard $x_b \geq 1$ on edges that leave the root, in order to separate the two observations. □

In situations like those of Example 5.3, we should try to obtain more information about $\mathcal{A}$ to resolve the choice between the two guards, by asking more membership queries for clocked words that are very similar to w_c and w'_c, hoping to find an adjacent pair (u_c, u'_c) of clocked words such that $\mathit{diff}(u_c, u'_c) \subset \mathit{diff}(w_c, w'_c)$, where $\subset$ denotes strict set inclusion. We formalize which clocked words are "very similar" to w_c by defining $\mathit{neighbourhood}(w_c)$ as the set of clocked words w'_c such that (w_c, w'_c) is an adjacent pair. Letting $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$, this means that $\mathit{neighbourhood}(w_c)$ is the set of clocked words $(a_1, \gamma'_1) \dots (a_n, \gamma'_n)$ that satisfy, for every $a \in \Sigma$ and $1 \leq i \leq n$,

- if $\gamma_i(x_a) \leq K$ is an integer then $\gamma_i(x_a) - 1 < \gamma'_i(x_a) < \gamma_i(x_a) + 1$, and
- if $\gamma_i(x_a) \leq K$ is non-integer or $\gamma_i(x_a) > K$ then $\gamma'_i(x_a) \approx_K \gamma_i(x_a)$.

We can then define a "closest" adjacent pair as a *critical pair*, used to infer guards in the tree, by the following definition.

Definition 5.4 *Let w_g be a guarded word. A* critical pair *in w_g is an adjacent pair (w_c, w'_c) of clocked words in w_g which are distinguished by Obs, such that there is no other adjacent pair (u_c, u'_c) of clocked words in w_g and in $\mathit{neighbourhood}(w_c)$, which are distinguished by Obs, such that $\mathit{diff}(u_c, u'_c) \subset \mathit{diff}(w_c, w'_c)$.* □

Intuitively, a critical pair is a pair of clocked words which are distinguished by Obs, and are "as adjacent as possible".

Example 5.5 Let $w_g = (a, x_a \geq 1 \wedge x_a < 2)(b, x_a \leq 2 \wedge x_b \geq 3)$, and let the pair (w_c, w'_c) be distinguished by Obs, where

$$w_c = (a, \left(\begin{smallmatrix} x_a \to 1.0 \\ x_b \to 1.0 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 2.0 \\ x_b \to 3.0 \end{smallmatrix} \right)) \text{ and}$$
$$w'_c = (a, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 1.5 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 3.0 \end{smallmatrix} \right))$$

Here, $\mathit{diff}(w_c, w'_c) = \{(1, x_a, 1, \leq), (1, x_b, 1, \leq), (2, x_a, 2, \geq)\}$. Since $|\mathit{diff}(w_c, w'_c)| > 1$, we search for clocked words in $\mathit{neighbourhood}(w_c)$ and w_g which are different from w_c and w'_c. Assume that we find

$$u_c = (a, \left(\begin{smallmatrix} x_a \to 1.6 \\ x_b \to 1.6 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 2.0 \\ x_b \to 3.6 \end{smallmatrix} \right)) \text{ and}$$
$$u'_c = (a, \left(\begin{smallmatrix} x_a \to 1.6 \\ x_b \to 1.6 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 3.1 \end{smallmatrix} \right))$$

Note that any other $v_c \models w_g$ is such that either $v_c \approx_K w_c$ or $v_c \approx_K w'_c$ or $v_c \approx_K u_c$, or $v_c \approx_K u'_c$. Note that the pairs (w_c, w'_c), (u_c, u'_c), (w_c, u_c), (w'_c, u'_c), and (w_c, u'_c) are adjacent pairs, but (w'_c, u_c) is not an adjacent pair. Now, if $Obs(u_c) \neq Obs(u'_c)$ then $\mathit{diff}(u_c, u'_c) = \{(2, x_a, 2, \geq)\}$, hence (u_c, u'_c) is a critical pair in w_g. If $Obs(u_c) = Obs(u'_c)$, then, on the one hand, if $Obs(u_c) = Obs(w'_c)$ (implying $Obs(w_c) \neq Obs(u_c)$ and $Obs(w_c) \neq Obs(u'_c)$) it follows from $\mathit{diff}(w_c, u_c) \subset \mathit{diff}(w_c, u'_c)$ and $|\mathit{diff}(w_c, w'_c)| = |\mathit{diff}(w_c, u_c)|$ that (w_c, w'_c) and (w_c, u_c) are critical pairs in w_g. On the other hand, if $Obs(u_c) = Obs(u'_c)$ and $Obs(u_c) = Obs(w_c)$, we obtain (observing that $\mathit{diff}(w'_c, u'_c) \not\subset \mathit{diff}(w_c, w'_c)$) that (w_c, w'_c) and (w'_c, u'_c) are critical pairs in w_g. □

5.2 Separating guards

The point of a critical pair is that it allows a natural construction of a guard that splits an inconsistent node or some of its ancestors, as described by the following definition.

Definition 5.6 *Let (w_c, w'_c) be a critical pair in w_g, An atomic guard $x_a \leq k$ is a* separating guard *at position i* inferred from (w_c, w'_c) *if*

- $(i, x_a, k, \leq) \in \mathit{diff}(w_c, w'_c)$ *and*
- *there is no $(j, x_b, k', \geq)$ or $(j, x_b, k', \leq)$ in $\mathit{diff}(w_c, w'_c)$ such that*
  - $j < i$, *or*
  - $j = i$ *and* $k' < k$.

Similarly, from the dual case the separating guard $x_a \geq k$ can be inferred. □

The definition states that if $\mathit{diff}(w_c, w'_c)$ has several elements, then priority is given to the guard at the earliest possible position, and if there is still a choice to resolve it in favour of the smallest constant. A reason for choosing the earliest difference is that such a split still remains in the tree, even if later a split is introduced into the tree which splits one of its ancestors; thus splits are never "undone".

In Example 5.2, we have $\mathit{diff}(w_c, w'_c) = \{(2, x_b, 1, \leq)\}$, whence (w_c, w'_c) is a critical pair and the separating guard $x_b \leq 1$ at position 2 is inferred from it. In Example 5.3, $\mathit{diff}(w_c, w'_c) = \{(1, x_a, 1, \geq), (1, x_b, 1, \geq)\}$. Since x_a and x_b have the same value at the first position, there are no closer adjacent pairs in the neighborhood of w_c. Hence (w_c, w'_c) is critical pair, and there are two possible separating guards, $x_a \geq 1$ and $x_b \geq 1$ at position 1. Definition 5.6 can still not give a priority between the two guards, so in this case both $x_a \geq 1$ and $x_b \geq 1$ are separating guards at position 1 inferred from the critical pair. In cases like this, we have to choose one of the inequalities as a guard in the tree. This choice can influence the size of the inferred automaton, as illustrated in Section 8.

Definition 5.6 describes how to create guards from critical pairs. Since our algorithm attempts to find precisely the guards that occur in $\mathcal{A}$, the automaton to be learned, we require that any guard be introduced only "by need", meaning that it is inferred from a critical pair. This is formalized as the property of being *well-guarded*.

Definition 5.7 *An observation structure $\langle Obs, \mathcal{N} \rangle$ is* well-guarded *if for each conjunct g_i that occurs in the last guard of some node $u_g(a, g_1 \wedge \ldots \wedge g_k)$ in $\mathcal{N}$ there is a critical pair (w_c, w'_c) which passes u_g, from which g_i, or its negation, is inferred at position i, where $i = |u_g| + 1$.* □

Intuitively, well-guardedness requires that each conjunct in a guard of the tree be motivated by a critical pair.

5.3 Finding Critical Pairs

In this subsection, we describe how to find a critical pair, when the timed decision tree contains an inconsistent node w_g, so that w_g or one of its ancestors can be split to resolve the inconsistency. The search procedure is an adaptation of binary search, and described in the proof of the following theorem.

Theorem 5.8 *Let $\langle Obs, \mathcal{N} \rangle$ be an observation structure. If there is an inconsistent node $w_g \in \mathcal{N}$, then a critical pair in w_g can be found by $O(m|\Sigma| \log K + 3^{m|\Sigma|})$ membership queries, where m is the length of w_g.*

Proof. Assume that the node w_g is inconsistent, i.e., there exist two clocked words w_c and w'_c, such that

- $w_c \vDash w_g$ and $w'_c \vDash w_g$,
- $Obs(w_c) \neq Obs(w'_c)$.

That is, w_c and w'_c lead to the same node, one is accepted and one is rejected. Our goal is to find a critical pair in w_g.

Our first step is to construct two adjacent clocked words u_c and u'_c leading to w_g such that one is accepted and one is rejected. We will use binary search on clocked words. First we define the notion of convex combination of clocked words. For two clocked words $v_c = (a_1, \gamma_1) \dots (a_n, \gamma_m)$ and $v'_c = (a_1, \gamma'_1) \dots (a_n, \gamma'_m)$ and two real numbers $\lambda_1, \lambda_2 \in \mathbb{R}^{\geq 0}$ with $\lambda_1 + \lambda_2 = 1$, define $\lambda_1 v_c + \lambda_2 v'_c$ to be the clocked word $(a_1, \gamma''_1) \dots (a_n, \gamma''_m)$ where $\gamma''_i(x_a) = \lambda_1 \gamma_i(x_a) + \lambda_2 \gamma'_i(x_a)$ for all $1 \leq i \leq m$ and $a \in \Sigma$. The binary search algorithm manipulates two clocked words u_c, u'_c that will be moved closer and closer to each other. Initially, $u_c = w_c$ and $u'_c = w'_c$. In each iteration we construct $u''_c = 0.5u_c + 0.5u'_c$. If $u''_c \not\approx_K u_c$ and $u''_c \not\approx_K u'_c$ a membership query for u''_c is performed. If $Obs(u_c) \neq Obs(u''_c)$ we set u'_c to the value of u''_c, otherwise we set u_c to the value of u''_c. We continue the binary search either until $\gamma_i(x_a) > K$ and $\gamma'_i(x_a) > K$ or until $-1 < \gamma_i(x_a) - \gamma'_i(x_a) < 1$ for all $1 \leq i \leq m$ and $a \in \Sigma$ where (a_i, γ_i) is the ith pair in u_c and (a_i, γ'_i) is the ith pair in the u'_c.

Assume that there are i and a such that the difference between values of clock x_a at position i in w_c and w'_c is much bigger than K. Assume that there are j and b such that the difference between values of clock x_b at position j in w_c and w'_c is smaller than K. Then after $\lceil \log K \rceil$ membership queries binary search construct u_c and u'_c such that the values of clock x_b at position j in u_c and u'_c are close to each other, but difference between values of clock x_a at position i in u_c and u'_c can be still bigger than K. To make values of clock x_a at position i close to each other we also need at most $\lceil \log K \rceil$ since some clocked words u''_c which binary search constructs will be such that $u''_c \approx_K u_c$ or $u''_c \approx_K u'_c$ and we do not need to perform a membership query for u''_c. It follows that for every $1 \leq i \leq m$ and $a \in \Sigma$ at most $\lceil \log K \rceil$ membership queries are performed. Then binary search performs at most $m|\Sigma| \lceil \log K \rceil$ membership queries.

Intuitively u_c and u'_c are "close", but they need not be adjacent. There can be i, b and $k \leq K$, $k \in \mathbb{N}$ such that

$$\gamma_i(x_b) > k \quad \textit{and} \quad \gamma'_i(x_b) < k$$

Then we construct the clocked word u''_c as a convex combination of u_c and u'_c such that (a_i, γ''_i) is ith pair in u''_c and $\gamma''_i(x_b) = k$. It can be done by setting $u''_c = \lambda u_c + (1-\lambda)u'_c$, where $\lambda = (k - \gamma'_i(x_b))/(\gamma_i(x_b) - \gamma'_i(x_b))$. Then a membership query for u''_c is performed. If $Obs(u''_c) \neq Obs(u_c)$ then we set $u'_c = u''_c$ otherwise we set $u_c = u''_c$. We repeat this process for u_c and u'_c until we find u_c and u'_c such that for every i and b there is a $k \leq K$ such that

- $\gamma_i(x_b) \approx_K \gamma'_i(x_b)$, or
- $\gamma_i(x_b) = k$ and $k - 1 < \gamma'_i(x_b) < k + 1$, or
- $\gamma'_i(x_b) = k$ and $k - 1 < \gamma_i(x_b) < k + 1$.

The number of membership queries performed at this step is at most $m|\Sigma|$.

It can be that u_c and u'_c are still not adjacent, since there are i, j, a, b and $k \leq K$, $k' \leq K$ such that $\gamma_i(x_b) = k$, $\gamma'_i(x_b) \neq k$, $\gamma_j(x_a) \neq k'$ and $\gamma'_j(x_a) = k'$. We can solve this problem by setting $u''_c = 0.5u_c + 0.5u'_c$. As result we get that u''_c is adjacent to u_c and u'_c. Then we perform membership query for u''_c. Assume $Obs(u_c) \neq Obs(u''_c)$ and set $u'_c = u''_c$.

Our second step is, starting with u_c and u'_c, to construct a critical pair. The critical pair can be found in $\textit{neighbourhood}(u_c)$ by performing at most $3^{m|\Sigma|}$ membership queries.

Then the number of membership queries that the algorithm performs is $O(m|\Sigma| \log K + 3^{m|\Sigma|})$. $\square$

5.4 Restructuring a Timed Decision Tree

In this subsection, we describe our procedure for restructuring a timed decision tree to remove an inconsistent node, after a new critical pair has been found to resolve the inconsistency. The restructuring will affect the subtree rooted at the node which is split by the guard inferred from the critical pair. Thus, the restructuring procedure should produce a new subtree which is well-guarded in the sense of Definition 5.7, and so that for all critical pairs (w_c, w'_c), where both w_c and w'_c pass the root of the reconstructed subtree, the clocked words w_c and w'_c lead to different nodes.

Our procedure is represented by the function $\textit{reconstruct\_subtree}(u_g)$, presented in Algorithm 1, which returns a restructured subtree rooted at u_g. It invokes the function *find_guards* to generate guards on the edges that lead to children of u_g. The function *find_guards* infers atomic guards from the relevant critical pairs, and organizes them into a set of conjunctions. In order to guarantee that splits are never "undone", we consider critical pairs in the same ordering as they were added to $Dom(Obs)$. For each generated child of u_g,

Algorithm 1 Procedure for constructing subtree from critical pairs

```
Function reconstruct_subtree(u_g) ←–
  for each a ∈ Σ such that Dom(Obs) contains a clocked word which passes u_g(a, true)
      for each guard g in find_guards(u_g, (a, true))
        add u_g(a, g) to N
        reconstruct_subtree(u_g(a, g))
end

Function find_guards(u_g, (a, g)) ←–
  let i ← |u_g(a, g)|
  if Obs contains no critical pair which passes u_g(a, g) from which a guard at position i
     can be inferred
    return {g}
  else
    let g' ← guard inferred at position i from a critical pair which passes u_g(a, g)
    return   find_guards(u_g, (a, g ∧ g')) ∪ find_guards(u_g, (a, g ∧ ¬g'))
end
```

reconstruct_subtree(u_g) invokes itself recursively as long as there are relevant observations in Obs.

That *reconstruct_subtree* performs its intended function is stated by the following proposition.

Proposition 5.9 *Given a partial mapping Obs and a guarded word u_g, the function reconstruct_subtree(u_g) produces a well-guarded subtree rooted at u_g, such that for all critical pairs (w_c, w_c'), where both w_c and w_c' pass u_g, the clocked words w_c and w_c' lead to different nodes.*

In particular, *reconstruct_subtree*(λ) constructs a timed decision tree $\mathcal{N}$ so that $\langle Obs, \mathcal{N}\rangle$ is well-guarded, and so that no critical pair is the cause of an inconsistency.

Proof. Well-guardedness follows by observing that *find_guards* only infers atomic guards from critical pairs. The second condition follows by noting that *find_guards* returns only when there are no critical pairs that remain to be separated. □

We can use the function *reconstruct_subtree*(u_g) to remove an inconsistent node as follows. Assume that to the observation structure $\langle Obs, \mathcal{N}\rangle$ we have added a critical pair in w_g from which the guard g' is inferred at position i. Let $u_g(a, g)$ be the prefix of length i of w_g. We then remove the subtree rooted at $u_g(a, g)$ and replace it by two new subtrees, one rooted at $u_g(a, g \wedge g')$, and the other rooted at $u_g(a, g \wedge \neg g')$. The first subtree is constructed by adding $u_g(a, g \wedge g')$ and then invoking *reconstruct_subtree*$(u_g(a, g \wedge g'))$, and the second by adding $u_g(a, g \wedge \neg g')$ and then invoking *reconstruct_subtree*$(u_g(a, g \wedge \neg g'))$.

If, after reconstruction, the tree is still inconsistent (note that the reconstruction does not guarantee absence of inconsistent nodes for which there is no critical pair), then we resolve it by again asking more membership queries in order to find a critical pair and thereafter again reconstruct the tree to remove the inconsistent node. This process can be repeated many times, but the num-

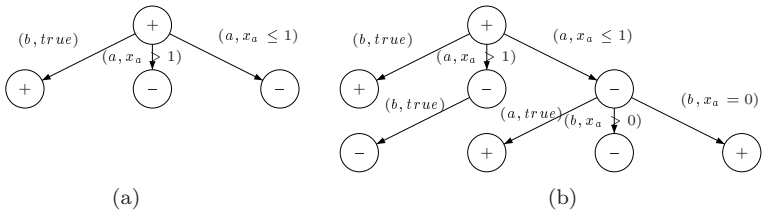


Figure 2: Introducing a split into the tree

ber of repetitions is bounded, since there is a bound on the number of possible guarded words of a given depth.

Example 5.10 We show an example of how a split is introduced in the tree. Recall the observation structure $\langle Obs, \mathcal{N} \rangle$ shown in Figure 1. The node $(a, true)(b, x_a = 0)$ is inconsistent. In order to resolve the inconsistency we need to find a critical pair. Suppose that the critical pair is

$$u_c = (a, \left(\begin{smallmatrix} x_a \to 1.0 \\ x_b \to 1.0 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.0 \\ x_b \to 1.0 \end{smallmatrix} \right)),$$
$$u'_c = (a, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 1.5 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.0 \\ x_b \to 1.5 \end{smallmatrix} \right))$$

We add u_c, u'_c to $Dom(Obs)$. There are two separating guards at position 1 which can be inferred from it. We choose the guard $x_a \leq 1$. Then we remove $(a, true)$ and all its descendants from $\mathcal{N}$ and add $(a, x_a \leq 1)$ and $(a, x_a > 1)$ to $\mathcal{N}$, see Figure 2(a).

There is no critical pair which passes $(a, x_a > 1)(b, true)$ or $(a, x_a \leq 1)(a, true)$. Since $(a, 1.5)(b, 1.5) \in Dom(Obs)$ passes $(a, x_a > 1)(b, true)$ and $(a, 0)(a, 0) \in Dom(Obs)$ passes $(a, x_a \leq 1)(a, true)$ then we add $(a, x_a > 1)(b, true)$ and $(a, x_a \leq 1)(a, true)$ to $\mathcal{N}$.

There is a critical pair

$$w_c = (a, \left(\begin{smallmatrix} x_a \to 0.5 \\ x_b \to 0.5 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.0 \\ x_b \to 0.5 \end{smallmatrix} \right)),$$
$$w'_c = (a, \left(\begin{smallmatrix} x_a \to 0.5 \\ x_b \to 0.5 \end{smallmatrix} \right))(b, \left(\begin{smallmatrix} x_a \to 0.1 \\ x_b \to 0.6 \end{smallmatrix} \right))$$

in $Dom(Obs)$ which passes $(a, x_a \leq 1)(b, true)$. Since the separating clock guard $x_a = 0$ at position 2 is inferred from (w_c, w'_c), we add $(a, x_a \leq 1)(b, x_a = 0)$ and $(a, x_a \leq 1)(b, x_a > 0)$ to $\mathcal{N}$, see Figure 2(b). Thereafter $\langle Obs, \mathcal{N} \rangle$ is a well-formed and consistent observation structure. □

6 Constructing Automata from Observation Structures

In this section, we describe how to fold a timed decision tree into a hypothesized TDERA, by merging nodes which have "similar" future behavior according to Obs. Since the construction is not trivial, we will first provide some motivation and an illustrating example.

6.1 Informal Overview

As motivation, let us consider the simpler case of untimed automata, which can be regarded as TDERAs in which all guards are *true*. When performing regular inference for untimed automata, following the principles used in e.g., [Ang87, Gol67, KV94, RS93, BDG97], we can first construct a timed decision tree with all guards being *true*, which is essentially a mapping from a prefix-closed set $Dom(Obs)$ of untimed words in Σ^* to $\{+, -\}$. To fold this mapping into an automaton, we find a prefix-closed subset U of $Dom(Obs)$, such that for all words $u \in U$ and alphabet symbols $a \in \Sigma$, the extension ua is in $Dom(Obs)$, and furthermore if ua is not in U then ua can be merged with some word u' in U. Since we want the resulting automaton to agree with Obs, this merging should be done only if the observed future behavior from ua is included in the observed future behavior from u', i.e., whenever $uaz \in Dom(Obs)$ for some suffix $z \in \Sigma^*$, then $u'z \in Dom(Obs)$ and $Obs(u'z) = Obs(uaz)$.

If in addition we require that for any two words u, u' in U, there is some suffix z that separates them, i.e., $uz, u'z \in Dom(Obs)$ and $Obs(uz) \neq Obs(u'z)$, then we can prove that the set U, which forms the locations of the resulting hypothesized automaton $\mathcal{H}$, will have at most as many members as there are locations in the automaton $\mathcal{A}$ to be learned [Ang87]. Namely, define a mapping θ from U to locations in $\mathcal{A}$ by $\theta(u) = l$ if the word u takes $\mathcal{A}$ from its initial location to l. We note that if u and u' are two different words in U, then $\theta(u) \neq \theta(u')$ since u and u' are separated by some suffix. Hence U cannot have more words than there are locations of $\mathcal{A}$. This bound on $\mathcal{H}$ can then be used to prove that the inference algorithm terminates [Ang87].

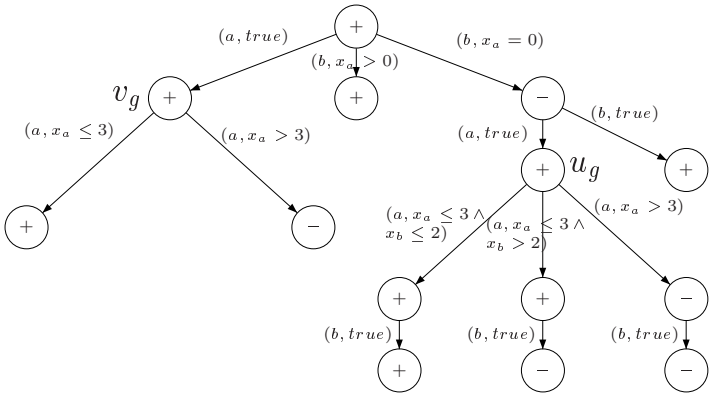


Figure 3: Illustration of inclusion between subtrees

Let us now consider how these principles can be transferred to the inference of TDERAs. A main question is to define the requirements for when a node (i.e., a guarded word) $u(a, g)$ in a timed decision tree can be merged with another node u'. It is natural to require that the subtree rooted at $u(a, g)$ be "included" in the subtree rooted at u'. Consider, e.g., the timed decision tree in Figure 3.

Here, the subtree rooted at the node $v_g = (a, true)$ to the left accepts all clocked suffix words of form (a, γ) such that $\gamma(x_a) \leq 3$, and rejects all clocked suffix words of form (a, γ) such that $\gamma(x_a) > 3$. This information is also included in the subtree rooted at the node $u_g = (b, x_a = 0)(a, true)$ to the right, hence we say that the subtree rooted at v_g is "included" in the subtree rooted at u_g, and allow v_g to be merged with u_g (for clarity, we have omitted edges from u_g and v_g for the symbol b). In Theorem 6.5, we establish that with inclusion between subtrees as the criterion for merging guarantees that the resulting hypothesized automaton $\mathcal{H}$ agrees with Obs.

A problem with the above merging requirement is that we cannot imitate the termination proof of the untimed case, in which separating suffixes play a crucial role to establish a bound on the size of constructed automata. The reason is that even if two nodes have different subtrees which cannot be merged, it may not be possible to find a separating suffix. To understand why this is the case, note that each node in the tree has a postcondition which restricts the suffixes of the node. For instance, in Figure 3 the nodes u_g and v_g have the postcondition $x_a \leq x_b$, which implies that, e.g., any suffix of form (a, γ) which can occur in a membership query must satisfy $\gamma(x_a) \leq \gamma(x_b)$. This shows that in general, two subtrees may disagree on the acceptance of suffixes that cannot occur in queries, implying that their roots cannot be separated by a suffix. A

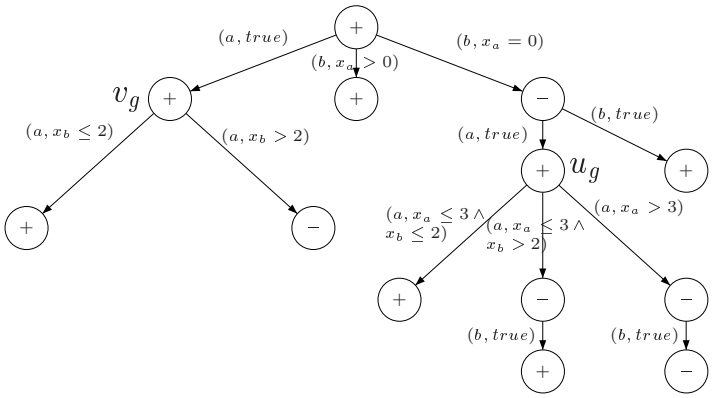


Figure 4: Illustration of alternative merging criterion

concrete example is found in Figure 4. Here, the subtrees rooted at the nodes v_g and u_g disagree on the acceptance of, e.g., the clocked suffix word (a, γ) where $\gamma(x_a) = 4$ and $\gamma(x_b) = 2$. However, this suffix cannot occur in a query since it does not satisfy the postcondition $x_a \leq x_b$ at v_g and u_g.

A way to imitate the termination argument in the untimed case is to define a different criterion for merging nodes, which requires that the two nodes have region-equivalent postconditions, and that their subtrees agree on suffixes that can actually occur in queries. As an illustration, consider the timed decision tree in Figure 4. Here, the subtree rooted at the node v_g to the left accepts all

clocked suffix words of form (a, γ) such that $\gamma(x_b) \leq 2$, and rejects all clocked suffix words of form (a, γ) such that $\gamma(x_b) > 2$. Thus, the node v_g cannot be merged with the node u_g according to the previous criterion. However, if we restrict attention to suffixes that satisfy the postcondition $x_a \leq x_b$, then the subtree rooted at u_g accepts all clocked suffix words of form (a, γ) such that $\gamma(x_b) \leq 2$, and rejects all clocked suffix words of form (a, γ) such that $\gamma(x_b) > 2$, since any suffix of form (a, γ) which satisfies the postcondition $x_a \leq x_b$ and $\gamma(x_a) > 3$ also satisfies $\gamma(x_b) > 2$.

With the new merging criterion, we can indeed require that two nodes in U which are not merged and have region-equivalent postconditions, must be separated by some suffix. A drawback of this new relation, however, is that nodes with postconditions which are not region-equivalent cannot be merged. In contrast, the first criterion, based on inclusion between subtrees, allows to merge nodes with postconditions which are not region-equivalent, often resulting in significantly smaller automata. In order to generate automata that are always of bounded size and often small, we therefore present a restructuring operation on trees, called "copying", with the goal that nodes which can be merged by the second criterion can also be merged by the first. Intuitively, the copying operation copies the structure of one subtree to that of another. For instance, in Figure 4, after copying the structure of the subtree rooted at u_g to the subtree rooted at v_g, we obtain the tree in Figure 5. After this operation, the subtree rooted at v_g is included in the subtree rooted at u_g. In general, the copying

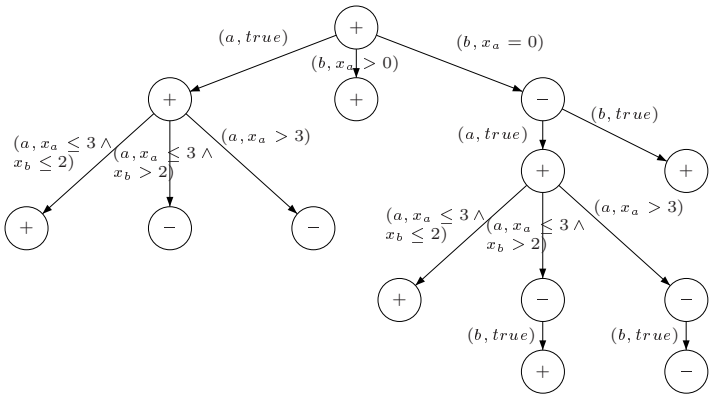


Figure 5: Result after copying operation

of nodes is nontrivial since it is recursive. For instance, it could be that a descendant of u_g has its structure copied from some other node.

To summarize, our procedure for folding a timed decision tree into a TDERA consists in first finding a prefix V of nodes, which allows an automaton to be formed by merging nodes according to the second criterion, so that we can prove a bound on the size of V. Thereafter, we perform the copying operation, and try to obtain a smaller TDERA by merging nodes according to the first criterion.

A precondition for this procedure is that the timed decision tree is well-formed and consistent. The bound on V further requires that sufficiently many queries are performed to have separating suffixes wherever needed. We call this latter property *region-consistency*.

This section is organized as follows. After presentation of a general operation for folding a tree into an automaton in the next section, in Section 6.3, we present the first merging criterion, based on "inclusion" between subtrees. In Section 6.4, we present the second construction, based on suffixes of observations. Section 6.5 describes the copying operation. In Section 7, we put all the pieces together by presenting the overall algorithm, and finally, in Section 7.1, we prove termination and give complexity bounds.

6.2 Preliminaries

Let $\mathcal{N}$ be a timed decision tree, and let $u_g, v_g \in \mathcal{N}$. Define $untimesuff_{\mathcal{N}}(u_g)$ as the set of (untimed) words z such that $\mathcal{N}$ contains a descendant $u_g z_g$ of u_g with $z = untime(z_g)$. Let $height(u_g, \mathcal{N})$ denote the height of the subtree rooted at u_g in $\mathcal{N}$. If $\leq$ is a preorder on nodes in $\mathcal{N}$, we use $v_g < u_g$ to denote that $v_g \leq u_g$ and $u_g \not\leq v_g$. We say that $\leq$ is *height-monotone* if $v_g \leq u_g$ implies $height(v_g, \mathcal{N}) \leq height(u_g, \mathcal{N})$. For a set U of nodes of $\mathcal{N}$, let $v_g \leq U$ denote that $v_g \leq u_g$ for some $u_g \in U$, and let $succ(U)$ denote the set of children of nodes in U which are not themselves in U.

Definition 6.1 *Let $\mathcal{N}$ be a timed decision tree, and let $\leq$ be a preorder on nodes in $\mathcal{N}$, A prefix-closed subset U of $\mathcal{N}$ is called*

- complete *if for all $u_g \in U$ and $a \in \Sigma$ there is some node $u_g(a, g) \in \mathcal{N}$,*
- $\leq$-closed *if $v_g \leq U$ for all $v_g \in succ(U)$,*
- $\leq$-unique *if U does not contain two nodes u_g, u'_g with $u_g \leq u'_g$.* □

Intuitively, if a prefix-closed subset U of $\mathcal{N}$ is complete and $\leq$-closed, then $\mathcal{N}$ can be folded into a TDERA by merging each node v_g in $succ(U)$ with some $u_g \in U$ such that $v_g \leq u_g$. The property of $\leq$-uniqueness will be used to bound the size of the resulting TDERA.

In the following subsections, we will present several constructions of an automaton from a timed decision tree, in which the locations of the automaton are a prefix-closed set of nodes of the tree. The following lemma gives conditions under which the existence of such a prefix-closed set of nodes can be guaranteed.

Lemma 6.2 *Let $\leq$ be a height-monotone preorder on nodes in $\mathcal{N}$. Then there exists a $\leq$-closed and $\leq$-unique prefix-closed subset of $\mathcal{N}$.*

Proof. Construct a set U of nodes of $\mathcal{N}$ by starting from the set $\{\lambda\}$ containing only the root, and repeatedly adding to U some successor v_g of a node already in U which satisfies

- $v_g \not\leq U$, and
- if $\mathcal{N}$ contains a node u'_g with $v_g < u'_g$, then u'_g has an ancestor v'_g in $succ(U)$ with $v'_g \leq U$.

We claim that when the addition of nodes to U cannot be continued, then U is prefix-closed, $\leq$-closed and $\leq$-unique.

- By construction, U is prefix-closed.
- To see that U is $\leq$-unique, note that when a node v_g is added to U, then U does not contain any u_g with $v_g \leq u_g$; furthermore, if U would contain some u_g with $v_g > u_g$, then at the time when u_g was added, v_g would have an ancestor u'_g in $succ(U)$ with $u'_g \leq U$, implying that u'_g and hence v_g will never be added to U.
- Finally, if U is not $\leq$-closed at termination, then there exist one or several nodes v_g in $succ(U)$ such that $v_g \not\leq U$. Each such node v_g satisfies (since it cannot be added) $v_g < u'_g$ for some node u'_g in $\mathcal{N} \setminus U$, which has a unique ancestor v'_g in $succ(U)$ with $v'_g \not\leq U$. The node v'_g in its turn satisfies (for the same reason) $v'_g < u''_g$ for some node u''_g in $\mathcal{N} \setminus U$, which must have an ancestor v''_g in $succ(U)$ with $v''_g \not\leq U$. Continuing in this way, we must eventually construct a sequence of nodes $v_g^0\ v_g^1\ v_g^2 \cdots v_g^n$ with $v_g = v_g^0$ which has a cycle, i.e., $v_g^k = v_g^n$ for some $k < n$. However, if the strict ancestor relation is used somewhere in the cycle from v_g^k to v_g^n, then we have $height(v_g^k, \mathcal{N}) < height(v_g^n, \mathcal{N})$ which contradicts $v_g^k = v_g^n$, and if the strict ancestor relation is never used in the cycle from v_g^k to v_g^n, then we have $v_g^k < v_g^{k+1} < \cdots < v_g^n$, which also contradicts $v_g^k = v_g^n$. □

We observe that the proof also provides a natural method for constructing a $\leq$-closed and $\leq$-unique prefix-closed subset of $\mathcal{N}$.

Given a certain prefix U of nodes of a timed decision tree $\mathcal{N}$, we can construct an ERA as follows.

Definition 6.3 *Let $\langle Obs, \mathcal{N} \rangle$ be a consistent observation structure, let $\leq$ be a height-monotone preorder on nodes in $\mathcal{N}$, and let U be a $\leq$-unique, $\leq$-closed, and prefix-closed subset of $\mathcal{N}$. Then a $U_{\leq}$-*merging *of $\langle Obs, \mathcal{N} \rangle$ is an ERA $\langle U, \{\lambda\}, L^f, E \rangle$, such that*

- $L^f = \{\ u_g \in U\ :\ Obs(u_g) = +\ \}$, *and*
- *for each node $u_g(a, g) \in \mathcal{N}$ with $u_g \in U$ there is exactly one edge in E of form $(u_g, u'_g, a, g) \in E$ with $u_g(a, g) \leq u'_g$.*

We observe that if, in addition, U is complete, then $\langle Obs, \mathcal{N} \rangle$ is a TDERA. □

Intuitively, the locations of a $U_{\leq}$-merging are the nodes in U. "Edges" between nodes in U are also edges of the automaton, since (by $\leq$-uniqueness of U) for an edge of form (u_g, u'_g, a, g) the node u'_g must be $u_g(a, g)$ if $u_g(a, g) \in U$. In addition, each "edge" from a node u_g in U to a node $u_g(a, g)$ in $succ(U)$ becomes an edge in the automaton from u_g to a node u'_g with $u_g(a, g) \leq u'_g$.

6.3 Constructing an Agreeing Automaton

In this section, we specialize the construction of Definition 6.3 by defining a relation, denoted $\leq_{un}$, between nodes. Intuitively $v_g \leq_{un} u_g$ defines the first merging criterion described in Section 6.1, and means that the subtree rooted at node v_g is "included" in the subtree rooted at node u_g, implying that v_g can be merged with u_g. We then prove that by using $\leq_{un}$ as a criterion for merging nodes, the result is a TDERA which agrees with Obs.

Intuitively, when comparing nodes to see whether they can be merged, we compare their overlapping suffixes, since these represent clocked words that pass the nodes. Two guarded words w_g and w'_g are *suffix-overlapped* if there is a clocked suffix word w_c such that $w_c \vDash w_g$ and $w_c \vDash w'_g$.

Definition 6.4 *Let $\langle Obs, \mathcal{N} \rangle$ be a consistent observation structure. A node $v_g \in \mathcal{N}$ is* unifiable *with a node $u_g \in \mathcal{N}$, denoted $v_g \leq_{un} u_g$, if*

- $untimesuff_{\mathcal{N}}(v_g) \subseteq untimesuff_{\mathcal{N}}(u_g)$, *and*
- *for any descendants $v_g z_g$ of v_g and $u_g z'_g$ of u_g with $Obs(v_g z_g) \in \{+, -\}$ such that z_g and z'_g are suffix-overlapped, we have $Obs(v_g z_g) = Obs(u_g z'_g)$.* □

It follows from Definition 6.4 that $\leq_{un}$ is height-monotone, transitive, and hence a preorder. Hence by Lemma 6.2 there exists a $\leq_{un}$-closed and $\leq_{un}$-unique prefix-closed subset U of $\mathcal{N}$, from which we can construct an $U_{\leq_{un}}$-merging. The following theorem states that this $U_{\leq_{un}}$-merging agrees with Obs.

Theorem 6.5 *Let $\langle Obs, \mathcal{N} \rangle$ be a consistent observation structure. Let U be a complete, $\leq_{un}$-unique, and $\leq_{un}$-closed prefix-closed subset of $\mathcal{N}$. Let $\mathcal{H}$ be a $U_{\leq_{un}}$-merging of $\langle Obs, \mathcal{N} \rangle$. Then for every clocked word $w_c \in Dom(Obs)$, if $Obs(w_c) = +$ then w_c is accepted by $\mathcal{H}$, otherwise w_c is rejected.*

Proof. Let $w_c \in Dom(Obs)$ and $Obs(w_c) = +$. Then there is a guarded word $w_g \in \mathcal{N}$ with $Obs(w_g) = +$ such that $w_c \vDash w_g$. If $w_g \in U$ then the lemma trivially holds. If $w_g \notin U$, then $w_g = u_g z_g$ for some $u_g \in U$ and suffix z_g of w_g. We prove by induction over the length of the guarded word z_g that whenever $u_g \in U$ and $Obs(u_g z_g) = +$ then each clocked suffix word z_c such that $z_c \vDash z_g$ is in $\mathcal{L}^+_c(\mathcal{H}, u_g)$.

The base case, where $z_g = \lambda$, follows trivially as above. Otherwise we can decompose z_g as $z_g = v_g z'_g$, where v_g is nonempty and $u_g v_g \in succ(U)$. By Definition 6.3, using that U is $\leq_{un}$-complete, there is a $u'_g \in U$ such that $u_g v_g \leq_{un} u'_g$. Let z'_c be the suffix of z_c with $|z'_c| = |z'_g|$. By Definition 6.4 there is a descendant $u'_g z''_g$ of u'_g such that $z'_c \vDash z''_g$, i.e., z'_g and z''_g are suffix-overlapped, and $Obs(u'_g z''_g) = +$. By the induction hypothesis (since z''_g is shorter than z_g) z'_c is in $\mathcal{L}^+_c(\mathcal{H}, u'_g)$. Hence z_c is in $\mathcal{L}^+_c(\mathcal{H}, u_g)$ (since $u_g \overset{v_g}{\Longrightarrow} u'_g \overset{z''_g}{\Longrightarrow} l'$ in $\mathcal{H}$ for some accepting location l').

The case $Obs(w_c) = -$ is analogous. □

6.4 Constructing Automata of Bounded Size

The construction in the preceding subsection produces, by Theorem 6.5, a TDERA which agrees with Obs. However, it seems difficult to use this construction to establish a bound the size of the constructed automaton, which is an important step in proving that the *Learner* eventually infers a TDERA equivalent to $\mathcal{A}$, after a bounded number of membership and equivalence queries. A main reason is that the relation $\leq_{un}$ compares nodes by the form of their respective subtrees. Since the number of different possible subtrees is unbounded, we can get an unbounded number of incomparable nodes. In this section, we present another relation for comparing nodes, which states that two nodes with region-equivalent postconditions can be merged if they are not separated by some suffix. This will allow to prove a bound on the constructed automata, as described in the informal overview of Section 6.1.

A precondition for our construction is that the observation structure contains enough queries to satisfy two conditions:

- queries for clocked words that pass some node should, when possible, use the same clocked word as prefix,
- each such prefix should have queries for enough suffixes to make it possible to compare nodes.

To define these conditions, we partition the clocked words that lead to some node u_g into equivalence classes, such that two clocked words are equivalent if they can be extended with equivalent suffixes.

Definition 6.6 *For two clocked words u_c and u'_c, define $u_c \simeq_K u'_c$ if for each clocked suffix word z_c such that $u_c z_c$ is a clocked word, there is a clocked suffix word z'_c with $z_c \approx_K z'_c$ such that $u'_c z'_c$ is a clocked word, and vice versa.* □

The following lemma characterizes $\simeq_K$ in terms of region equivalence.

Lemma 6.7 *Let u_c and u'_c be two clocked words. Then $u_c \simeq_K u'_c$ if and only if either $u_c = u'_c = \lambda$ or $\gamma[x_a \to 0] \sim_K \gamma'[x_{a'} \to 0]$, where (a, γ) and (a', γ') is the last pair in u_c and u'_c, respectively.*

Intuitively, $u_c \simeq_K u'_c$ if u_c and u'_c lead to the same initial region.

Proof. We first establish the if-direction. Let u_c and u'_c be as in the statement of the lemma, and assume that $\gamma[x_a \to 0] \sim_K \gamma'[x_{a'} \to 0]$, where (a, γ) and (a', γ') is the last pair in u_c and u'_c, respectively. By Definition 6.6, we must prove that for each clocked suffix word z_c such that $u_c z_c$ is a clocked word, there is a clocked suffix word z'_c with $z_c \approx_K z'_c$ such that $u'_c z'_c$ is a clocked word. The case $z_c = \lambda$ is trivial. To prove the case $|z_c| = 1$, let (b, ρ) be a pair consisting of a symbol $b \in \Sigma$ and clock valuation ρ such that $u_c(b, \rho)$ is a clocked word. We will prove the claim that there is a clock valuation ρ' with $\rho \approx_K \rho'$ such that $u'_c(b, \rho')$ is a clocked word and $\rho[x_b \to 0] \sim_K \rho'[x_b \to 0]$. We can then use

this claim to prove the lemma for the general case by induction: simply add one pair, consisting of a symbol and a clock valuation, at a time.

To prove the claim, we must, by the definition of $\sim_K$ in Section 2, establish that

A for all $x_c \in C_\Sigma$, either

- $\rho(x_c)$ and $\rho'(x_c)$ are both greater than K, or
- $\lfloor \rho(x_c) \rfloor = \lfloor \rho'(x_c) \rfloor$ and $fract(\rho(x_c)) = 0$ iff $fract(\rho'(x_c)) = 0$,

and

B for all $x_c, x_{c'} \in C_\Sigma$ where c, c' differ from b, with $\rho(x_c) \leq K$ and $\rho(x_{c'}) \leq K$,
$fract(\rho(x_c)) \leq fract(\rho(x_{c'}))$ iff $fract(\rho'(x_c)) \leq fract(\rho(x_{c'}))$.

The condition on clocked suffix words in Section 2 states that $\rho(x_c) = \rho(x_a) + \gamma(x_c)$ for all $x_c \in C_\Sigma$ except x_a. This means that ρ is determined from $\rho(x_a)$, and that we must just find an appropriate value of $\rho'(x'_a)$ to determine ρ'. Let d be the symbol in Σ such that $\rho(x_d) \leq K$, and such that $\rho(x_d)$ has the least fractional part among all clocks c with $\rho(x_c) \leq K$. Choose a value of $\rho'(x'_a)$ such that $\rho'(x_d) \leq K$, and such that $\rho'(x_d)$ has the least fractional part among all clocks c with $\rho'(x_c) \leq K$. Then condition A follows from $\gamma[x_a \to 0] \sim_K \gamma'[x_{a'} \to 0]$. Condition B also follows from $\gamma[x_a \to 0] \sim_K \gamma'[x_{a'} \to 0]$, since differences between clock valuations are preserved for all clocks except x_b.

To prove the only if-direction, assume that $\gamma[x_a \to 0] \not\sim_K \gamma'[x_{a'} \to 0]$. We shall find a pair (b, ρ) consisting of a symbol $b \in \Sigma$ and clock valuation ρ such that $u_c(b, \rho)$ is a clocked word, but there is no clock valuation ρ' with $\rho \approx_K \rho'$ such that $u'_c(b, \rho')$ is a clocked word or vice versa. If there is some clock c such that $\gamma[x_a \to 0](x_c) \not\approx_K \gamma'[x_{a'} \to 0](x_c)$, then if $\gamma[x_a \to 0](x_c) < \gamma'[x_{a'} \to 0](x_c)$ we can simply choose b as c and ρ as $\gamma[x_a \to 0]$ Otherwise, if for instance $fract(\gamma[x_a \to 0](x_c)) < fract(\gamma[x_a \to 0](x_{c'}))$ but $fract(\gamma'[x_{a'} \to 0](x_c)) \geq fract(\gamma[x_{a'} \to 0](x_{c'}))$ we can find a constant $k \leq K$ and ρ such that $\rho(x_c) < k < \rho(x'_c)$, for which there is no equivalent ρ'. □

We can now start to formalize what it means for an observation structure to contain enough queries.

Definition 6.8 *Let $\langle Obs, \mathcal{N} \rangle$ be an observation structure, let $u_g \in \mathcal{N}$, and let $[\gamma']_K$ be an initial region such that $[\gamma']_K \cap sp(u_g) \neq \emptyset$. A clocked word $u_c(a, \gamma)$ in u_g is a* region-representative of $[\gamma']_K$ in u_g *if $\gamma[x_a \to 0] \sim_K \gamma'$ and for every $u'_c z'_c \in Dom(Obs)$ with $u'_c \vDash u_g$ and $u_c(a, \gamma) \simeq_K u'_c$, there is a $u_c(a, \gamma) z_c \in Dom(Obs)$ such that $z_c \approx_K z'_c$.* □

Intuitively, a region-representative $u_c(a, \gamma)$ of $[\gamma']_K$ in u_g has the property that all queries which pass through u_g, hitting the same initial region as $u_c(a, \gamma)$, are also performed with $u_c(a, \gamma)$ as prefix. We use the convention that if u_g is the root of the tree then λ is a region-representative of the (only) initial region $[\gamma]_K$

in u_g, which satisfies $\gamma(x_a) = 0$ for all $a \in \Sigma$. Note that we do not require that $u_c(a, \gamma) \in Dom(Obs)$.

The following definition formalizes the two properties which together say that an observation structure contains enough queries, stated at the beginning of this subsection.

Definition 6.9 *An observation structure* $\langle Obs, \mathcal{N} \rangle$ *is* region-consistent *if*

- *for every node* $u_g \in \mathcal{N}$ *and initial region* $[\gamma']_K$ *such that* $[\gamma']_K \cap sp(u_g) \neq \emptyset$ *there is a region-representative* u_c *of* $[\gamma']_K$ *in* u_g*, and*
- *for any two nodes* v_g *and* u_g *with* $sp(v_g) \approx_K sp(u_g)$ *and* $height(v_g, \mathcal{N}) \leq height(u_g, \mathcal{N})$ *we have*
  - *for every initial region* $[\gamma']_K$ *with* $[\gamma']_K \cap sp(u_g) \neq \emptyset$*, the region-representative* v_c *of* $[\gamma]_K$ *in* v_g *and the region-representative* u_c *of* $[\gamma]_K$ *in* u_g *have the property that for each* z_c *such that* $v_c z_c \in Dom(Obs)$ *there is a* y_c *with* $y_c \approx_K z_c$ *such that* $u_c y_c \in Dom(Obs)$*.* □

Definition 6.10 *Let* $u_g z_g$ *and* $v_g z'_g$ *be two guarded words. We say that the suffix* z_g *of* u_g overlaps *with the suffix* z'_g *of* v_g *if there are* $u_c z_c$ *and* $v_c z'_c$ *such that* $u_c z_c \vDash u_g z_g$*,* $v_c z'_c \vDash u_g z'_g$ *and* $z_c \approx_K z'_c$*.* □

We introduce a property of a timed decision tree which we call suffix-completeness, the aim of which is to reduce the number of equivalence queries. It checks, for two nodes that do not agree on overlapping suffixes, that there is indeed some observation that separates them to make sure that the nodes cannot be merged.

Definition 6.11 *An observation structure* $\langle Obs, \mathcal{N} \rangle$ *is* suffix-complete *if for every* $v_g, u_g \in \mathcal{N}$ *with* $height(v_g, \mathcal{N}) \leq height(u_g, \mathcal{N})$ *we have*

- $untimesuff_{\mathcal{N}}(v_g) \subseteq untimesuff_{\mathcal{N}}(u_g)$ *and*
- *whenever the suffix* z_g *of* u_g *overlaps with the suffix* z'_g *of* v_g*, and* $Obs(v_g z_g) \neq Obs(u_g z'_g)$*, then there are clocked words* $v_c z_c, u_c z'_c \in Dom(Obs)$ *with* $v_c z_c \vDash v_g z_g$ *and* $u_c z'_c \vDash u_g z'_g$ *such that* $z_c \approx_K z'_c$*.*

□

We can now define the second relation of Section 6.1 between nodes, denoted $\preceq_{un}$, which implies that two nodes can be merged if they have region-equivalent postconditions and agree on suffixes that occur in performed queries.

Definition 6.12 *Let* $\langle Obs, \mathcal{N} \rangle$ *be a well-formed, consistent, and region-consistent observation structure. Define the relation* $\preceq_{un}$ *between nodes by* $v_g \preceq_{un} u_g$ *if*

- $sp(v_g) \approx_K sp(u_g)$*,*
- $untimesuff_{\mathcal{N}}(v_g) \subseteq untimesuff_{\mathcal{N}}(u_g)$*, and*

- *for any descendants $v_g z_g$ of v_g and $u_g z'_g$ of u_g such that the suffix z_g of v_g overlaps with the suffix z'_g of u_g, we have $Obs(v_g z_g) = Obs(u_g z'_g)$.* □

Intuitively, $v_g \preceq_{un} u_g$ means that all observations represented by the subtree rooted at v_g are also represented by the subtree rooted at u_g. It follows from Definition 6.12 that $\preceq_{un}$ is height-monotone, transitive, and hence a preorder. Note that $\preceq_{un}$ differs from $\leq_{un}$ in that it compares only the parts of subtrees that satisfy the postcondition of the respective nodes: in this way inclusion between subtrees is checked only for the parts in which observations are possible. The following lemma states that comparable nodes must in fact agree on suffixes of performed queries.

Lemma 6.13 *Let $\langle Obs, \mathcal{N} \rangle$ be a well-formed, consistent, and region-consistent observation structure. If $v_g \preceq_{un} u_g$ then whenever $v_c z_c \in Dom(Obs)$ for some $v_c \models v_g$, there is a y_c with $y_c \approx_K z_c$ such that $u_c y_c \in Dom(Obs)$ for some $u_c \models u_g$ and $Obs(u_c y_c) = Obs(v_c z_c)$.*

Proof. The existence of y_c follows by region consistency, and $Obs(u_c y_c) = Obs(v_c z_c)$ follows from $v_g \preceq_{un} u_g$. □

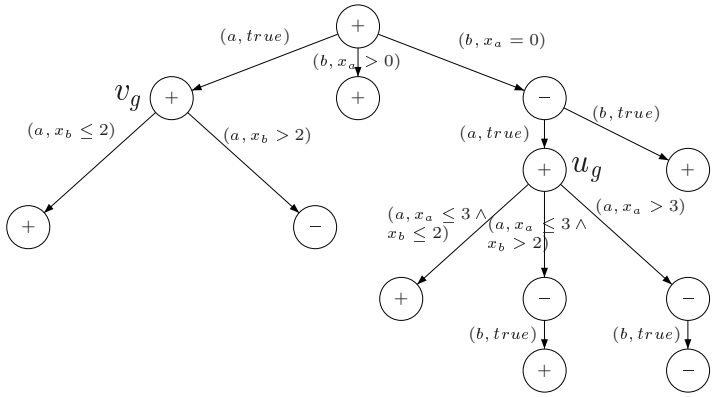
Using Lemma 6.13, we can now obtain a bound on the size of a $\preceq_{un}$-unique set of nodes in a timed decision tree, by adapting the corresponding argument for the L^*-algorithm. This bound allows to prove that if U is a complete, $\preceq_{un}$-unique, and $\preceq_{un}$-closed prefix-closed subset of $\mathcal{N}$, then there is a bound on size of a $U_{\leq_{un}}$-merging. A disadvantage of using the relation $\preceq_{un}$ when constructing automata, however, is that nodes with postconditions which are not region-equivalent will not be merged. In contrast, the relation $\leq_{un}$ allows to merge nodes with postconditions which are not region-equivalent, often resulting in significantly smaller automata.

In order to generate automata that are always of bounded size and often small, we therefore present an operation on timed decision trees, called *copying*. The purpose of this operation is to restructure the timed decision tree $\mathcal{N}$ of an observation structure $\langle Obs, \mathcal{N} \rangle$ so that there exists a $\leq_{un}$-closed and $\leq_{un}$-unique prefix of $\mathcal{N}$ which is also $\preceq_{un}$-unique. This will allow us to prove the same upper bound on the size of $U_{\leq_{un}}$-mergings as on the size of $U_{\preceq_{un}}$-mergings.

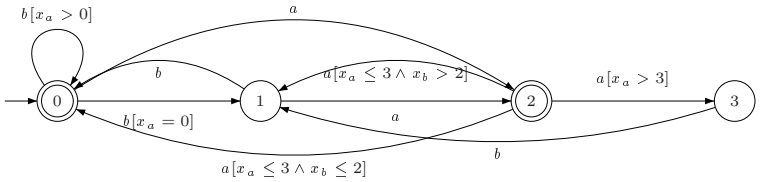
We define two nodes to be *incompatible* if they are separated by some suffix.

Definition 6.14 *Let $\langle Obs, \mathcal{N} \rangle$ be an observation structure. Two guarded words, u_g and v_g, in $\mathcal{N}$ are* incompatible*, denoted $u_g \not\asymp_K v_g$, if $Dom(Obs)$ contains clocked words $u_c z_c$ and $v_c z'_c$ with $u_c \models u_g$ and $v_c \models v_g$, such that $z_c \approx_K z'_c$ and $Obs(u_c z_c) \neq Obs(v_c z'_c)$.* □

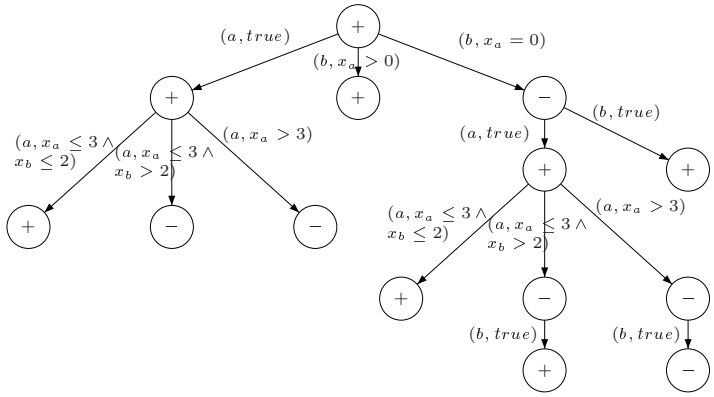
We write $u_g \asymp_K v_g$ to denote that u_g and v_g are not incompatible (i.e., compatible).



(a) $\mathcal{N}$



(b) $\mathcal{N}_{\preceq un}$



(c) $\mathit{unfold}_{\mathcal{N}}(\mathcal{N}_{\preceq un})$

Figure 6: Copying subtrees

6.5 The Copying Operation

In this section we describe the *copying* operation which is performed before the construction in Theorem 6.5, in order to guarantee a bounded size on the resulting automaton. The copying operation restructures the timed decision tree $\mathcal{N}$ of an observation structure $\langle Obs, \mathcal{N} \rangle$ so that there exists a $\leq_{un}$-closed and $\leq_{un}$-unique prefix of $\mathcal{N}$ which is also $\preceq_{un}$-unique. The construction assumes that we have found a $\preceq_{un}$-closed and $\preceq_{un}$-unique prefix V of $\mathcal{N}$. This means that for any node v_g in $succ(V)$, there is a u_g in V with $v_g \preceq_{un} u_g$. However, it is not guaranteed that there is a u_g in V with $v_g \leq_{un} u_g$. The goal of the copying operation is to restructure the subtree rooted at v_g by "copying" so that $v_g \leq_{un} u_g$ for some $u_g \in V$. As a result, V will become $\leq_{un}$-closed, and we can thereafter search for a $\leq_{un}$-closed and $\leq_{un}$-unique prefix-closed subset U of V, which by construction will also be $\preceq_{un}$-unique. This will allow us to prove the same upper bound on the size of $U_{\leq_{un}}$-mergings as on the size of $U_{\preceq_{un}}$-mergings.

Definition 6.15 *Let $\mathcal{N}_{\preceq_{un}}$ be a $V_{\preceq_{un}}$-*merging *of $\langle Obs, \mathcal{N} \rangle$. Define $unfold_{\mathcal{N}}(\mathcal{N}_{\preceq_{un}})$ to be the timed decision tree such that $w_g \in unfold_{\mathcal{N}}(\mathcal{N}_{\preceq_{un}})$ iff $\lambda \stackrel{w_g}{\Longrightarrow} l$ in $\mathcal{N}_{\preceq_{un}}$ for some location l in $\mathcal{N}_{\preceq_{un}}$, and there is some $w'_g \in \mathcal{N}$ with $untime(w_g) = untime(w'_g)$.* □

Intuitively, if we have found a $\preceq_{un}$-closed and $\preceq_{un}$-unique prefix-closed set of nodes V, then the copying operation folds $\mathcal{N}$ by merging nodes which are equivalent wrp. to $\preceq_{un}$, whereafter $\mathcal{N}$ is again unfolded. Note that V may contain leaves.

Example 6.16 To illustrate the copying operation, consider the timed decision tree shown in Figure 6(a). It has two nodes $v_g = (a, true)$ and $u_g = (b, x_a = 0)(a, true)$ such that $sp(v_g)$ and $sp(u_g)$ are both $x_a \leq x_b$. Since $Obs(v_g(a, x_b \leq 2)) \neq Obs(u_g(a, x_a > 3))$ and suffixes $(a, x_b \leq 2)$ and $(a, x_a > 3)$ are suffix-overlapped, it follows that $v_g \not\leq_{un} u_g$. On the other hand, if we restrict the subtrees rooted at u_g and v_g to suffixes that can actually be part of clocked words, then they do not disagree, i.e., $v_g \preceq_{un} u_g$. In order to make v_g unifiable with u_g, the copying procedure copies the subtree rooted at u_g to the subtree rooted at v_g, obtaining the timed decision tree shown in Figure 6(c). □

The following lemma implies that after the copying operation preserves consistency of a well-formed tree, implying that after copying, we can use Theorem 6.5 to construct a TDERA that agrees with Obs.

Lemma 6.17 *Let $\langle Obs, \mathcal{N} \rangle$ be a well-formed and consistent observation structure. Let V be a $\preceq_{un}$-unique and $\preceq_{un}$-closed prefix-closed set of nodes in $\mathcal{N}$. Construct $\mathcal{N}' = unfold_{\mathcal{N}}(\mathcal{N}_{\preceq_{un}})$ as in Definition 6.15. Then $\langle Obs, \mathcal{N}' \rangle$ is a consistent observation structure.*

Proof. It follows directly that nodes in $V \cup succ(V)$ are consistent. We prove, using induction over the length of the guarded word z_g, that whenever $v_g \in V \cup succ(V)$, then for any descendants $v_g y_g, v_g y'_g \in \mathcal{N}$ and $v_g z_g \in \mathcal{N}'$ such that the suffix z_g of v_g overlaps with the suffix y_g of v_g and such that the suffix z_g of v_g overlaps with the suffix y'_g of v_g, we have $Obs(v_g y_g) = Obs(v_g y'_g)$, and hence $v_g z_g$ must be a consistent node.

We first consider the case $v_g \in succ(V)$. Let us use the notation $(v_g, \mathcal{N})$ to stress that we consider a node v_g which belongs to the tree $\mathcal{N}$. The case where $z_g = y_g = y'_g = \lambda$ follows trivially. So, assume that $z_g \neq \lambda$. If $y_g = y'_g$, we are done. Otherwise, by Definition 6.3 there is $u_g \in V$, such that $(v_g, \mathcal{N}) \preceq_{un} (u_g, \mathcal{N})$, i.e., $sp(v_g) \approx_K sp(u_g)$ and v_g is merged with u_g when constructing $\mathcal{N}_{\preceq_{un}}$. Then either

- $u_g z_g \in V \cup succ(V)$, in which case the claim follows by observing that $(v_g, \mathcal{N}) \preceq_{un} (u_g, \mathcal{N})$ implies that $Obs(v_g y_g) = Obs(u_g z_g) = Obs(v_g y'_g)$, since from $sp(v_g) \approx_K sp(u_g)$ we have that the suffix z_g of u_g overlaps with the suffix y_g of v_g and the suffix z_g of u_g overlaps with the suffix y'_g of v_g, or

- there is a prefix $u_g v'_g \in succ(V)$ of $u_g z_g$. Since the suffix y_g of v_g overlaps with suffix z_g of v_g and $sp(v_g) \approx_K sp(u_g)$ we infer that the suffix y_g of u_g overlaps with the suffix z_g of u_g. Then there is a node $u_g v'_g x_g \in \mathcal{N}$ such that the suffix $v'_g x_g$ of u_g overlaps with the suffix y_g of v_g and the suffix $v'_g x_g$ of u_g overlaps with the suffix z_g of u_g, from which we use $(v_g, \mathcal{N}) \preceq_{un} (u_g, \mathcal{N})$ to infer $Obs(u_g v'_g x_g) = Obs(v_g y_g)$. In an analogous manner, we infer that there is a node $u_g v'_g x'_g \in \mathcal{N}$ such that the suffix $v'_g x'_g$ of u_g overlaps with the suffix y'_g of v_g and the suffix $v'_g x'_g$ of u_g overlaps with the suffix z_g of u_g, and $Obs(u_g v'_g x'_g) = Obs(v_g y'_g)$. Let $u_g z_g = u_g v'_g z'_g$. Since the suffix $v'_g x_g$ of u_g overlaps with the suffix z_g of u_g we infer that the suffix x_g of $u_g v'_g$ overlaps with the suffix z'_g of $u_g v'_g$. Moreover, since the suffix $v'_g x'_g$ of u_g overlaps with the suffix z_g of u_g we infer that the suffix x'_g of $u_g v'_g$ overlaps with the suffix z'_g of $u_g v'_g$. By the inductive hypothesis, since z'_g is shorter than z_g, we infer that $Obs(u_g v'_g x_g) = Obs(u_g v'_g x'_g)$. Then $Obs(v_g y_g) = Obs(v_g y'_g)$.

It is straightforward to use this claim to establish the claim for nodes $v_g \in V$. □

Lemma 6.18 *Let $\langle Obs, \mathcal{N} \rangle$ be a well-formed, consistent, and region-consistent observation structure. Let V be a $\preceq_{un}$-unique, and $\preceq_{un}$-closed prefix-closed set of nodes in $\mathcal{N}$. Let $\mathcal{N}_{\preceq_{un}}$ be a $V_{\preceq_{un}}$-merging of $\mathcal{N}$. Construct $\mathcal{N}' = unfold_{\mathcal{N}}(\mathcal{N}_{\preceq_{un}})$ as in Definition 6.15. Then V is $\leq_{un}$-closed in $\mathcal{N}'$.*

Proof. Let $\mathcal{N}_{\preceq_{un}} = \langle U, \{\lambda\}, L^f, E \rangle$. We prove that if $u_g \in V$, $v_g = u'_g(a, g) \in succ(V)$, $(v_g, \mathcal{N}) \preceq_{un} (u_g, \mathcal{N})$, and there is $(u'_g, u_g, a, g) \in E$ then $(v_g, \mathcal{N}') \leq_{un} (u_g, \mathcal{N}')$. Namely, then for every node $v_g z_g \in \mathcal{N}'$ it follows, from Lemma 6.13, that there is a node $u_g z_g \in \mathcal{N}'$ such that $Obs(v_g z_g) = Obs(u_g z_g)$, from which we infer $(v_g, \mathcal{N}') \leq_{un} (u_g, \mathcal{N}')$. □

We remark that if V is complete in $\mathcal{N}$, then it is also complete in $\mathcal{N}'$. Copying can destroy well-formedness of the tree, but well-formedness is not required when constructing a TDERA in Definition 6.3.

From Lemma 6.18 it follows that there is a $\leq_{un}$-closed and $\leq_{un}$-unique prefix-closed subset U of V in $\mathcal{N}'$. Then U is also $\preceq_{un}$-unique since $U \subseteq V$. To make U complete, whenever $u_g \in U$ and $a \in \Sigma$ are such that there is no $u_g(a, g) \in \mathcal{N}'$, we add $u_g(a, true)$ to $\mathcal{N}'$ and ask membership query for u_c with $u_c \vDash u_g(a, true)$. Furthermore, the extra node $u_g(a, true)$ satisfies $u_g(a, true) \leq_{un} u'_g$ for some $u'_g \in U$ implying that the addition preserves $\leq_{un}$-closedness of U, which is required by Definition 6.3.

7 The Overall Algorithm

We can now finally bring all pieces together and present the overall algorithm for inferring a TDERA. The algorithm is represented by the function *Learner*, described in Algorithm 2.

Lines 2 – 3 perform initialization of the observation structure by constructing the initial timed decision tree with the empty word as root and children for each symbol in the alphabet. Lines 5 – 38 contain the steps for constructing a well-formed, consistent, region-consistent, and suffix-complete observation structure. Here lines 6 – 9 show the steps for achieving well-formedness, lines 10 – 14 show the steps for achieving consistency, lines 15 – 27 are concerned with achieving region-consistency and lines 28–37 are concerned with achieving suffix-completeness. After having constructed a well-formed, consistent, region-consistent and suffix-complete observation structure, lines 40 – 48 construct a hypothesis TDERA to prepare an equivalence query. Here line 40 finds a suitable prefix V, whereafter the copying operation is performed in lines 41 – 42. Lines 43-47 finds a suitable prefix U of V, and after fixing possible problems with U being not complete, the hypothesis $\mathcal{H}$ is formed in line 48. The equivalence query is posed in line 50. If it is successful, the algorithm terminates, otherwise the counterexample is inserted into the observation structure and the algorithm is restarted from line 5.

7.1 Termination proof and complexity results

In this section, we prove that our inference algorithm terminates after a bounded number of membership and equivalence queries. As a first step, we prove an upper bound on the number of locations in any constructed automaton.

Lemma 7.1 *Let $\langle Obs, \mathcal{N} \rangle$ be a well-formed, consistent, and region-consistent observation structure. Let V be a $\preceq_{un}$-unique set of nodes in $\mathcal{N}$. Then V has at most $|L + 1|^{R_K}$ nodes, where $|L|$ is the number of locations of $\mathcal{A}$ and R_K is the number of regions.*

Proof. Let the TDERA $\mathcal{A}$ be $\langle L, \{l^0\}, L^f, E \rangle$. Assign to each node $u_g \in V$ a mapping $\theta[u_g]$ from initial regions that intersect $sp(u_g)$ to $L \cup \{\perp\}$, as follows.

Algorithm 2 Inference of TDERA

```
1  Function Learner() ←−
2     let 𝒩 be {λ}
3     ask membership queries for λ and some (a, γ) for each a ∈ Σ
4
5     repeat
6        if there is a node w_g ∈ 𝒩 with Obs(w_g) = ⊥
7           ask a membership query for some w_c with w_c ⊨ w_g
8        if there is a clocked word w_c ∈ Dom(Obs) but no w_g ∈ 𝒩 with w_c ⊨ w_g
9           add appropriate nodes of form u_g(a_1, true)...(a_k, true) to 𝒩
10       if there is a node w_g ∈ 𝒩 with Obs(w_g) = ⊤
11          find a critical pair (w_c, w_c') in w_g
12          infer separating guard g at position i from (w_c, w_c')
13          let u_g be the prefix of w_g of length i − 1
14          replace the subtree rooted at u_g by reconstruct_subtree(u_g)
15       for each u_g ∈ 𝒩 and [γ]_K such that [γ]_K ∩ sp(u_g) ≠ ∅
16          if there is no region−representative of [γ]_K in u_g ∈ 𝒩
17             choose region−representative of [γ]_K in u_g
18          if u_c is region−representative of [γ]_K in u_g ∈ 𝒩
19             for each u_c'z_c' ∈ Dom(Obs) where u_c' ⊨ u_g and u_c ≃_K u_c'
20                make sure that there is z_c ≈_K z_c' with u_c z_c ∈ Dom(Obs)
21       if there are u_g, v_g ∈ 𝒩 with sp(u_g) ≈_K sp(v_g) and
22       height(v_g, 𝒩) ≤ height(u_g, 𝒩)
23          for every initial region [γ']_K with [γ']_K ∩ sp(u_g) ≠ ∅,
24             let v_c be the region−representative of [γ]_K in v_g
25             let u_c be the region−representative of [γ]_K in u_g
26             for each z_c such that v_c z_c ∈ Dom(Obs)
27                make sure that there is y_c ≈_K z_c with u_c y_c ∈ Dom(Obs)
28       for every u_g z_g and v_g z_g' such that the suffix z_g of u_g overlaps with
29       the suffix z_g' of v_g and Obs(u_g z_g) ≠ Obs(v_g z_g')
30             if there is no u_c z_c, v_c z_c' ∈ Dom(Obs) such that
31             u_c z_c ⊨ u_g z_g, v_c z_c' ⊨ v_g z_g', and z_c ≈_K z_c'
32                ask membership queries for u_c z_c and v_c z_c',
33                where u_c z_c ⊨ u_g z_g, v_c z_c' ⊨ v_g z_g', and z_c ≈_K z_c'
34       if there are u_g, v_g ∈ 𝒩 such that height(v_g, 𝒩) ≤ height(u_g, 𝒩),
35       v_g(a_1, g_1)...(a_n, g_n) ∈ 𝒩, u_g(a_1, g_1')...(a_{n−1}, g_{n−1}') ∈ 𝒩 and
36       there is no u_g(a_1, g_1')...(a_n, g_n') ∈ 𝒩
37          add u_g(a_1, g_1')...(a_{n−1}, g_{n−1}')(a_n, true) to 𝒩
38    until ⟨Obs, 𝒩⟩ is well−formed, consistent, region−consistent, and suffix−complete
39
40    let V be ⪯_un −closed and ⪯_un −unique prefix−closed subset of 𝒩
41    let 𝒩_⪯un be a V_⪯un −merging of ⟨Obs, 𝒩⟩
42    let 𝒩' be unfold_𝒩(𝒩_⪯un)
43    let U be a ≤_un −closed and ≤_un −unique prefix−closed subset of V
44    if U is not complete
45       for every u_g ∈ U and a ∈ Σ such that there is no u_g(a, g) ∈ 𝒩'
46             add u_g(a, true) to 𝒩'
47             ask a membership query for u_c with u_c ⊨ u_g(a, true)
48    let ℋ be a U_≤un −merging of ⟨Obs, 𝒩'⟩
49
50    ask an equivalence query for ℋ to Teacher
51    if the answer to the equivalence query is a counterexample w_c
52       goto line 5
53    else return ℋ
54 end
```

For an initial region $[\gamma]_K$ such that $[\gamma]_K \cap sp(u_g) \neq \emptyset$, let $\theta[u_g]([\gamma]_K) = l$ if there is a region-representative u_c of $[\gamma]_K$ in u_g, such that u_c leads to the location $l \in L$ in $\mathcal{A}$, i.e., such that $l^0 \stackrel{w_g}{\Longrightarrow} l$ in $\mathcal{A}$ for some w_g with $u_c \models w_g$. Let $\theta[u_g]([\gamma]_K) = \perp$ if $[\gamma]_K \cap sp(u_g) = \emptyset$. Note that region-consistency does not require that there is some $u_c y_c \in Dom(Obs)$, but still every u_c leads to some location in $\mathcal{A}$. We next observe that if $\theta[u_g]([\gamma]_K) = \theta[u'_g]([\gamma]_K) \neq \perp$ for an initial region $[\gamma]_K$, i.e., both u_g and u'_g have region-representatives of $[\gamma]_K$ which lead to the same location l, then whenever $u_c z_c, u'_c z'_c \in Dom(Obs)$ such that $u_c \models u_g$, $u'_c \models u'_g$, and $z_c \approx_K z'_c$, then $Obs(u_c z_c) = Obs(u'_c z'_c)$. It follows that if $\theta[u_g]$ and $\theta[u'_g]$ agree on all initial regions, then $sp(u_g) \approx_K sp(u'_g)$ and $u_g \asymp_K u'_g$. Since $\langle Obs, \mathcal{N} \rangle$ is region-consistent, then $u_g \preceq_{un} u'_g$ or $u'_g \preceq_{un} u_g$. Since V is $\preceq_{un}$-unique, the size of V is at most $|L+1|^{R_K}$ nodes. □

Theorem 7.2 *Algorithm 2 terminates.*

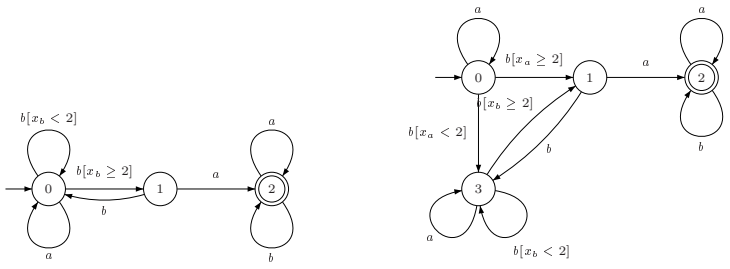
Proof. To prove termination of Algorithm 2 we should prove that the **repeat-until** loop in lines 5 – 38 terminates, and that there is a bound on the number of unsuccessful equivalence queries posed in line 50. Termination of the **repeat-until** loop follows by observing that no operation increases the depth of the tree, and that the number of possible timed decision trees of a given depth is bounded.

From Theorem 6.5 follows that every hypothesised automaton $\mathcal{H}$ which is constructed agrees with Obs. Thus every counterexample which is returned in an unsuccessful equivalence query is new and every constructed hypothesis $\mathcal{H}$ is different from any previously constructed hypothesis. From Lemma 7.1 follows that the size of $\mathcal{H}$ is bounded. Since the number of different TDERAs of bounded size is bounded, then the outer loop in function *Learner* terminates. □

A more precise (but still very large) bound on the number of queries is established in the following theorem.

Theorem 7.3 *The algorithm for learning TDERA performs* $O((\frac{(|\Sigma|+2K+1)e}{\Sigma})^{|\Sigma|(l+1)})$ *queries, where* l *is the length of the longest counterexample and* e *is the base of natural logarithm.*

Proof. An equivalence query introduces at least one new node into the tree. Since the postcondition defines at least an ordering on clocks, then a node can have at most $\binom{|\Sigma|+2K+1}{|\Sigma|}$ children. This is the number of non-increasing sequences of $|\Sigma|$ elements, where each element has values among 0 to $2K+1$. The height of the tree is at most $l+1$. From [Odl95] we know that $\binom{|\Sigma|+2K+1}{|\Sigma|} \leq (\frac{(|\Sigma|+2K+1)e}{\Sigma})^{|\Sigma|}$. It follows that the number of nodes in the final tree, and hence the number of equivalence queries is $O((\frac{(|\Sigma|+2K+1)e}{\Sigma})^{|\Sigma|(l+1)})$. A membership query is performed at most once for every simple guarded word of length at most $l+1$. Then the number of membership queries is $O((\frac{(|\Sigma|+2K+1)e}{\Sigma})^{|\Sigma|(l+1)})$. □



(a) $\mathcal{A}$ (b) $\mathcal{A}'$

Figure 7: Automata $\mathcal{A}$ and $\mathcal{A}'$

A bound on the parameter l in Theorem 7.3 can be obtained by assuming that returned counterexamples are as short as possible.

Theorem 7.4 *Let $|L|$ be the number of locations in $\mathcal{A}$. Assume that unsuccessful equivalence queries return a shortest (in number of symbols) counterexample. Then the length of any counterexample is less than $2|L|R_K$ where R_K is the number of regions.*

Proof. It was shown in [GJL05] that $\mathcal{A}$ can be transformed into an equivalent simple TDERA $\mathcal{A}'$ with at most $|L|R_K$ states. Then inference algorithm for deterministic finite automata can be applied to learn $\mathcal{A}'$. It follows from [TB73] that the length of counterexamples is less than $2|L|R_K$. □

For comparison, in [GJL05] is presented an algorithm that infers the region graph of some automaton equivalent to $\mathcal{A}$. The number of equivalence queries is at most $|L|R_K$ and the number of membership queries is $O(|\Sigma|^2|L|^2R_K^2lK)$, where R_K is the number of regions, l is the length of longest counterexample and L is the number of locations in $\mathcal{A}$.

8 Example

Suppose the automaton to be learned is the event-recording automaton $\mathcal{A}$ in Figure 7(a). We assume that we know that $K = 2$. We start by asking membership queries for λ, $(a, 0)$ and $(b, 0)$. Then we construct the timed decision tree $\mathcal{N}_1$ shown in Figure 8(a). Then $V = \{\lambda\}$ and $\mathcal{N}_1' = \mathcal{N}_1$, i.e. $\mathcal{N}_1$ is not changed after copying. Then $U = \{\lambda\}$ is $\leq_{un}$-closed, $\leq_{un}$-unique, complete and $\preceq_{un}$-unique prefix-closed subset of V in $\mathcal{N}_1'$. The Learner constructs hypothesized automaton $\mathcal{H}_1$ that is $U_{\leq_{un}}$-merging of $\langle Obs, \mathcal{N}_1' \rangle$, shown in Figure 8(b).

Assume that the counterexample $(b, 2)(a, 2.4)$ is returned. It is accepted by $\mathcal{A}$ but rejected by $\mathcal{H}_1$. Then we construct the tree $\mathcal{N}_2$, shown in Figure 9(a).

The observation structure $\langle Obs, \mathcal{N}_2 \rangle$ is not suffix-complete, since the suffix $(a, true)$ of λ overlaps with the suffix $(a, true)$ of $(b, true)$, $Obs((a, true)) \neq$

$Obs((b, true)(a, true))$, and there are no $u_c, u'_c z_c \in Dom(Obs)$ such that $u_c \vDash (a, true)$, $u'_c z_c \vDash (b, true)(a, true)$ and $u_c \approx_K z_c$. We ask membership query for $(b, 0)(a, 0)$. Since $(b, 0)(a, 0)$ is rejected by $\mathcal{A}$, then the node $(b, true)(a, true)$ becomes inconsistent. We perform binary search and ask membership queries for $(b, 1)(a, 1.2)$, $(b, 1.5)(a, 1.8)$ and $(b, 1.75)(a, 2.1)$. Then we have the critical pair

$$(b, \left(\begin{smallmatrix} x_a \to 2 \\ x_b \to 2 \end{smallmatrix} \right))(a, \left(\begin{smallmatrix} x_a \to 2.4 \\ x_b \to 0.4 \end{smallmatrix} \right)) \text{ and}$$
$$(b, \left(\begin{smallmatrix} x_a \to 1.75 \\ x_b \to 1.75 \end{smallmatrix} \right))(a, \left(\begin{smallmatrix} x_a \to 2.1 \\ x_b \to 0.35 \end{smallmatrix} \right))$$

from which we infer separating guards $x_a \geq 2$ and $x_b \geq 2$ at position 1. At this point we (arbitrarily) choose inequality $x_a \geq 2$ and construct timed decision tree $\mathcal{N}_3$, shown in Figure 9(b), by splitting node $(b, true)$. Then $\langle Obs, \mathcal{N}_3 \rangle$ is consistent. Since the suffix $(a, true)$ of λ overlaps with the suffix $(a, true)$ of $(b, x_a \geq 2)$, $Obs((a, true)) \neq Obs((b, x_a \geq 2)(a, true))$, and there are no u_c and $u'_c z_c$ such that $u_c \vDash (a, true)$, $u'_c z_c \vDash (b, x_a \geq 2)(a, true)$ and $u_c \approx_K z_c$ then $\langle Obs, \mathcal{N}_3 \rangle$ is not suffix-complete. Then we ask membership queries for $(a, 2.4)$ and $(b, 2)(a, 4.4)$.

Let $(b, 1.5)$ be a region-representative of $[\gamma]_K$ in $(b, x_a < 2)$, where $\gamma(x_a) = \gamma(x_b) = 1.5$. Then $\langle Obs, \mathcal{N}_3 \rangle$ is suffix-complete, but not region-consistent, since we have observations

$$u_c z_c = (b, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 1.5 \end{smallmatrix} \right))(a, \left(\begin{smallmatrix} x_a \to 1.8 \\ x_b \to 0.3 \end{smallmatrix} \right)) \text{ and}$$
$$u'_c z'_c = (b, \left(\begin{smallmatrix} x_a \to 1.75 \\ x_b \to 1.75 \end{smallmatrix} \right))(a, \left(\begin{smallmatrix} x_a \to 2.1 \\ x_b \to 0.35 \end{smallmatrix} \right))$$

where $u_c \neq \lambda$, $z_c \neq \lambda$, $u'_c \neq \lambda$, $z'_c \neq \lambda$ and $u_c \simeq_K u'_c$, but $z_c \not\approx_K z'_c$. Let

$$u_c z''_c = (b, \left(\begin{smallmatrix} x_a \to 1.5 \\ x_b \to 1.5 \end{smallmatrix} \right))(a, \left(\begin{smallmatrix} x_a \to 2.1 \\ x_b \to 0.6 \end{smallmatrix} \right))$$

Since $u_c z''_c \approx_K u'_c z'_c$ then $u_c z''_c$ is accepted iff $u'_c z'_c$ is accepted. Then we do not need to ask membership query for $u_c z''_c$. To make $\langle Obs, \mathcal{N}_3 \rangle$ region-consistent we add $u_c z''_c$ to $Dom(Obs)$.

We have $(b, x_a < 2)(a, true) \preceq_{un} (a, true)$. Then $V = \{\lambda, (a, true), (b, x_a < 2), (b, x_a \geq 2), (b, x_a \geq 2)(a, true)\}$ is $\preceq_{un}$-closed and $\preceq_{un}$-unique prefix-closed subset of $\mathcal{N}_3$. Then $\mathcal{N}'_3 = \mathcal{N}_3$, i.e. $\mathcal{N}_3$ is not changed after copying. To make U $\leq_{un}$-closed, we add $(b, x_a \geq 2)$ and $(b, x_a \geq 2)(a, true)$ to U. To make

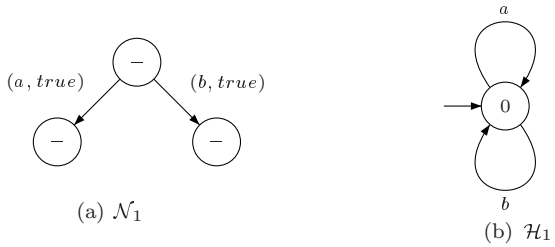
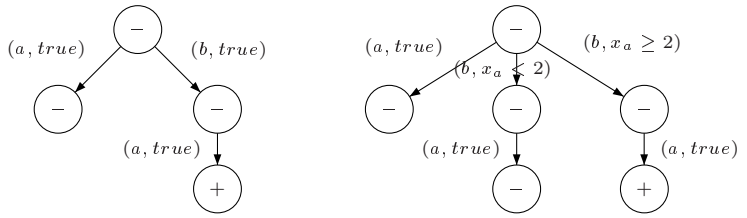
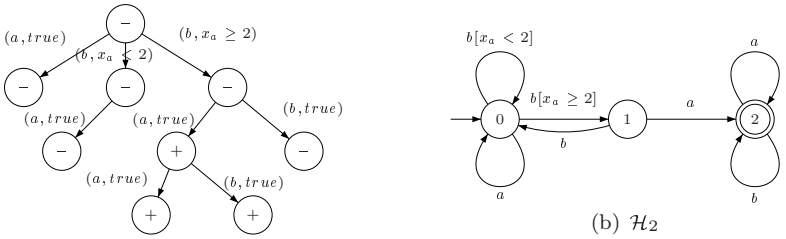


Figure 8: A tree $\mathcal{N}_1$ and an automaton $\mathcal{H}_1$



(a) $\mathcal{N}_2$ (b) $\mathcal{N}_3$

Figure 9: Trees $\mathcal{N}_2$ and $\mathcal{N}_3$



(a) $\mathcal{N}_3'$ (b) $\mathcal{H}_2$

Figure 10: A tree $\mathcal{N}_3'$ and an automaton $\mathcal{H}_2$

U complete we add $(b, x_a \geq 2)(a, true)(a, true)$, $(b, x_a \geq 2)(a, true)(b, true)$ and $(b, x_a \geq 2)(b, true)$ to $\mathcal{N}_3'$ and ask membership queries for $(b, 2)(b, 2)$, $(b, 2)(a, 2)(a, 2)$, and $(b, 2)(a, 2)(b, 2)$. The resulting timed decision tree $\mathcal{N}_3'$ is shown in Figure 10(a). Then U is $\leq_{un}$-closed, $\leq_{un}$-unique, complete and $\preceq_{un}$-unique prefix-closed subset of V in $\mathcal{N}_3'$.

The Learner construct hypothesized automaton $\mathcal{H}_2$ that is $U_{\leq_{un}}$-merging of $\langle Obs, \mathcal{N}_3' \rangle$, shown in Figure 10(b). The procedure continues and the Learner will construct three more automata before the automaton $\mathcal{A}'$, shown in Figure 7(b), is finally created.

9 Conclusion

We have presented a technique for inference of timed systems that can be represented as event-recording automata. We introduced timed decision tree as a data structure for organizing the results of the membership and equivalence queries. The timed decision tree is folded into an event-recording automaton by a merging procedure that is based on a notion of unifiable nodes. This is the first inference algorithm for the full class of event-recording automata which avoids explicit use of the region graph. The algorithm works under assumption

that a greatest constant K which appears in clock guards is known. To prove termination of the algorithm we introduced region-representatives. The number of region-representatives depends on K, and it is not clear how to prove termination of the algorithm if K is unknown.

The drawback of the algorithm that it has high complexity. However, the algorithm can be modified in such a way that it can learn DERAs with one clock by asking only a polynomial number of membership and equivalence queries. It would be interesting to find a class of DERAs with two clocks for which a smallest automaton can be learned by asking polynomial number of membership and equivalence queries. A difference between timed automata with one and two clocks is shown in [LMS04]. Laroussinie et al. present the polynomial algorithm for model checking $TCTL_{\leq,\geq}$ over timed automata with one clock and show that model checking CTL over timed automata with two clocks is PSPACE-complete.

Another interesting problem is to develop algorithm for learning deterministic timed automata. Difference between timed automata and DERAs that we do not know how many clocks a timed automaton has and they can reset at any transition. We can construct a timed decision tree by introducing at every transition a new clock, but then it is not clear how a merging procedure should work for constructing a timed automaton from the timed decision tree, since we need to compare edges which are labeled by guards containing different clocks.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH99] R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[BDG97] José L. Balcázar, Josep Díaz, and Ricard Gavaldá. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. Kluwer, 1997.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer-Verlag, 1998.

[BGJ+05] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In *Proc. of 8th*

International Conference on Fundamental Approaches to Software Engineering, pages 175–189, 2005.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *LNCS*, pages 232–243. Springer-Verlag, 1996.

[CDH+00] J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, and H. Zheng. Bandera : Extracting finite-state models from java source code. In *Proc. 22nd Int. Conf. on Software Engineering*, June 2000.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Dec. 1999.

[Cho78] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, pages 197–212, 1989.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, pages 313–329, 1998.

[FJJV97] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29, 1997.

[GJL04] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. In *Proceedings of the Joint Conferences FORMATS and FTRTFT*, volume 3253 of *LNCS*, September 2004.

[GJL05] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Inference of timed transition systems. *ENCS*, 138(3):87–99, 2005.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[GPY02] Alex Groce, Doron Peled, and Mihalis Yannakakis. Adaptive model checking. In *Proc. TACAS'02*, LNCS, 2002.

[HHNS02] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In R.-D. Kutsche and H. Weber, editors, *Proc. FASE02*, volume 2306 of *LNCS*, pages 80–95. Springer Verlag, 2002.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M.B. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Engineering*, 16(4):403–414, April 1990.

[HMP94] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112:173–337, 1994.

[HNS03] Hardi Hungar, Oliver Niese, and Bernhard Steffen. Domain-specific optimization in automata learning. In *Proc. CAV'03*, 2003.

[Hol00] G.J. Holzmann. Logic verification of ANSI-C code with SPIN. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification: Proc. 7^{th} Int. SPIN Workshop*, volume 1885 of *LNCS*, pages 131–147, Stanford, CA, 2000. Springer Verlag.

[HRS98] T.A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In K.G. Larsen, S. Skuym, and G. Winskel, editors, *ICALP98*, volume 1443 of *LNCS*, pages 580–591. Springer Verlag, 1998.

[KV94] M.J. Kearns and U.V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[LMS04] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of 15th International Conference on Concurrency Theory*, pages 387–401, 2004.

[MP04] O. Maler and A. Pnueli. On recognizable timed languages. In *Proc. FOSSACS04*, LNCS. Springer-Verlag, 2004.

[Odl95] A. M. Odlyzko. Asymptotic enumeration methods. In *Handbook of Combinatorics*, volume 2, pages 1063–1229. North-Holland, Amsterdam, 1995.

[RS93] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

[SEG00] M. Schmitt, M. Ebner, and J. Grabowski. Test generation with autolink and testcomposer. In *Proc. 2nd Workshop of the SDL Forum Society on SDL and MSC - SAM'2000*, June 2000.

[SV96] Jan Springintveld and Frits Vaandrager. Minimizable timed automata. In B. Jonsson and J. Parrow, editors, *Proc. FTRTFT'96*, volume 1135 of *LNCS*, pages 130–147. Springer Verlag, 1996.

[SVD01] Jan Springintveld, Frits W. Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theor. Comput. Sci.*, 254(1-2):225–257, 2001.

[TB73] B.A. Trakhtenbrot and J.M. Barzdin. *Finite automata: behaviour and synthesis*. North-Holland, 1973.

[Vas73] M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetic*, 9(4):653–665, 1973.

[VdWW06] Sicco E. Verwer, Mathijs M. de Weerdt, and Cees Witteveen. Identifying an automaton model for timed data. In *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*, pages 57–64. Benelearn, 2006.

[Wil94] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In H Langmaack, W. P. de Roever, and J. Vytopil, editors, *Proc. FTRTFT'94*, volume 863 of *LNCS*, pages 694–715. Springer Verlag, 1994.