Computer Science at Kent

Compositional Detection of Zeno Behavior in Timed Automata with Deadlines

Rodolfo Gómez

Technical Report No. 03-09
June 2009

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent, CT2 7NF, UK

Compositional Detection of Zeno Behavior in Timed Automata with Deadlines*

Rodolfo Gómez

Computing Laboratory, University of Kent, United Kingdom
R.S.Gomez@kent.ac.uk

June 30, 2009

Abstract

We present a compositional analysis to detect Zeno behavior in Timed Automata with Deadlines. A syntactic analysis is defined, based on Tripakis' strong non-Zenoness property, which identifies all elementary cycles where Zeno behavior may occur. This analysis is complemented by TCTL reachability properties, which characterize the occurrence Zeno behavior in potentially unsafe cycles.

1 Introduction

Timelocks [6, 7] are anomalous states in timed transition systems, where time divergence is prevented along all execution paths. Hence, timelocks may prevent the exploration of interesting behaviors (e.g., erroneous behaviors), which makes the verification of correctness properties unreliable.

Timed Automata with Deadlines (TAD) [18, 5, 6] are a variant of timed automata [2], where *deadlines* express the time progress conditions. Deadlines are clock constraints associated with transitions in the automaton, which determine when the transition must be executed. Importantly, neither internal actions nor synchronization on observable actions are made urgent unless they can be executed (TAD are *time-reactive* [4]). Hence, TAD avoid the most common form of timelocks, where neither actions nor delays may occur (*time-actionlocks* [6]). On the other hand, *Zeno behavior* may occur in TAD models, which refers to the anomalous behavior where actions occur infinitely often in finite time. Unfortunately, unlike time-actionlocks, Zeno behavior cannot be prevented at the semantic level.

Zeno behavior may adopt the form of *Zeno runs* or *Zeno-timelocks*. A *Zeno run* is an infinite run that is not time-divergent (i.e., a run where actions occur infinitely often in finite time). A Zeno timelock is a timelock, s, where a Zeno run exists from any state that is reachable from s. Zeno behavior compromises the verification of correctness properties, where the progress of time is inherently assumed. It is known that time divergence (and thus, the absence of Zeno behavior) can be characterized by liveness properties in TCTL [14] (e.g., see [22]). However, there is little tool support for the analysis of TAD models (the IF toolset [8], and the TAD2TA tool [10], are the notable exceptions), and the verification liveness properties (and in particular, time-divergence) is computationally expensive.

Our contribution. We offer an efficient method to guarantee the *absence of* Zeno behavior in TAD networks. The method is based on an improved, compositional application of Stripakis' original *strong*

*This research has been supported by the UK Engineering and Physical Sciences Research Council under grant EP/D067197/1.

non-Zenoness analysis [20]. The method is compositional and works on the syntax of network components, which makes it potentially more efficient than liveness analysis of time-divergence [14, 19]. Moreover, it provides the additional benefit of identifying *all* possible sources of Zeno behavior directly on the TAD network, which facilitates model debugging. The tradeoff, which is inherent to the analysis being applied at syntax level, is that the *occurrence of* Zeno behavior cannot be confirmed. False positives may be returned in the form of cycles in the component automata where Zeno behavior may (or may not) occur. In order to reduce, or even eliminate such false positives, we also derive simple TCTL reachability formulae, which characterize the occurrence of Zeno behavior in the offending cycles.

The combined syntactic and reachability analysis, as proposed in this paper, may potentially eliminate the need to perform liveness analysis. In addition, this approach can be easily implemented in verification tools for TAD models: the syntactic analysis works on the structure of automata, and the proposed TCTL reachability formulae express a simple class of safety properties (e.g., these can be verified in the IF toolset [8] and the TAD2TA tool [10]).

Related Work. Timelocks have been studied extensively in the literature of formal notations for timed systems, e.g. in timed process algebras [16, 17] and timed automata [14, 18, 5, 20, 6, 7, 11]. In particular, [20] and (more recently) [7, 11] provide detailed studies of Zeno behavior in timed safety automata [13]. Also, liveness analysis of time-divergence can be performed in a few model-checkers for timed automata, including Kronos [22], Uppaal [3] and RED [21]. However, to the best of our knowledge, this is the first time that Zeno behavior is studied on timed automata with deadlines, and the first attempt to propose feasible alternatives to liveness analysis.

The analysis presented in this paper is a compositional extension of our previous analysis of Zeno behavior on networks of timed safety automata [7], in the sense that the network's product automaton need not be constructed. In particular, compositionality is facilitated by the semantics of urgency in TAD.

Organization of this paper. Section 2 presents the formal syntax and semantics of Timed Automata with Deadlines, and defines Zeno behavior. Section 3 and 4 resp. present the compositional methods to detect (or rule out) Zeno behavior. Conclusions are given in Section 5.

2 Timed Automata with Deadlines

This section presents a common form of Timed Automata with Deadlines [18, 5, 6], where transitions are classified either as *lazy actions* (non-urgent), *eager actions* (urgent as soon as they are enabled), or *delayable actions* (urgent on their upper bounds). Formally, we will define the model using eager and lazy actions as the only primitives; delayable actions can be derived from these [5].

Preliminaries. Let $CA = \{a, b, \ldots\}$ and $HA = \{a?, a! \,|\, a \in CA\}$ (we define complementary labels, s.t. $\overline{a!} = a?$ and $\overline{a?} = a!$). Let $D = \{lazy, eager\}$. Let $\mathbb{C}$ be the set of clocks (variables that range in the non-negative reals, $\mathbb{R}^{+0}$). Let Φ be the set of clock constraints over $\mathbb{C}$, s.t.

$$\phi \in \Phi ::= true \,|\, x \sim c \,|\, x - y \sim c \,|\, \phi \wedge \phi$$

where $x, y \in \mathbb{C}$, $\sim \in \{<, >, =, \leq, \geq\}$ and $c \in \mathbb{N}$. A *valuation* is a mapping from $\mathbb{C}$ to $\mathbb{R}^{+0}$. Let $\mathbb{V}$ be the set of valuations. Let $\models$ denote the satisfiability of clock constraints over valuations. Let $v \in \mathbb{V}$, $\delta \in \mathbb{R}$ and $r \subseteq \mathbb{C}$. The valuation $v + \delta \in \mathbb{V}$ is defined s.t. $(v + \delta)(x) = v(x) + \delta$ if $v(x) + \delta \geq 0$, and $(v + \delta)(x) = 0$ otherwise, for all $x \in \mathbb{C}$. The valuation $r(v) \in \mathbb{V}$ is defined s.t. $r(v)(x) = 0$ for all $x \in r$, and $r(v)(x) = v(x)$ for all $x \notin r$.

Syntax and semantics. A *timed automaton with deadlines* (TAD) is a tuple of the form $A = (L, l_0, Lab, T, C)$, where L is a set of locations; $l_0 \in L$ is the initial location; $Lab \subseteq CA \cup HA$ is a set of labels; $T \subseteq L \times \Phi \times Lab \times D \times 2^{\mathbb{C}} \times L$ is a set of transitions (edges) and $C \in \mathbb{C}$ is a set of clocks.

Given a transition $t = (l, a, g, d, r, l') \in T$, l is the source location, a is the label; g is the guard; d is the deadline, r is the reset set and l' is the target location (resp., $src(t)$, $lab(t)$, $g(t)$, $d(t)$, $r(t)$ and $tgt(t)$). Transitions labeled with $a \in CA$ (resp. $a \in HA$) will be referred to as *completed actions* (resp. *half actions*). Transitions with deadline *lazy* (resp. *eager*) will be referred to as *lazy actions* (resp. *eager actions*).

A *TAD network* is a tuple $|A = \langle A_1, \ldots, A_n \rangle$, where $A_i = (L_i, l_{i,0}, Lab_i, T_i, C_i)$ is a TAD $(i : 1..n)$. Let $\mathbb{C} = \bigcup_{i=1}^{n} C_i$ (we say that $x \in \mathbb{C}$ is a *shared clock* if $x \in C_i \cup C_j$ for some $1 \leq i \neq j \leq n$; otherwise x is a *local clock*). The behavior of $|A$ is given by the timed transition system [15] $(S, s_0, Lab \cup \mathbb{R}^+, T)$, where $S \subseteq (\prod_{i=1}^{n} L_i) \times \mathbb{V}$ is the set of states (states are denoted $s = \langle \bar{l}, v \rangle$, where $\bar{l} \in \prod_{i=1}^{n} L_i$ and $v \in \mathbb{V}$); $s_0 \in S$ is the initial state ($s_0 = \langle \bar{l}_0, v_0 \rangle$, s.t. $\bar{l}_0 = \langle l_{1,0}, \ldots, l_{n,0} \rangle$ and $\forall\, x \in \mathbb{C}.\, v_0(x) = 0$); $Lab = CA$ is the set of action labels and $T \subseteq S \times Lab \cup \mathbb{R}^+ \times S$ is the smallest set of transitions that satisfies the following conditions.

(Transitions in T may be *action transitions*, (s, a, s') where $a \in Lab$, or *delay transitions*, (s, δ, s'), where $\delta \in \mathbb{R}^+$. We refer to elements of $\prod_{i=1}^{n} L_i$ as *location vectors*. We use $\bar{l}[l'_i/l_i]$ to denote substitution of l'_i for l_i in the location vector $\bar{l} = \langle l_1, \ldots, l_n \rangle$.)

1. **(completed actions)** $\left(\langle \bar{l}, v \rangle, a, \langle \bar{l}\,[l'_i/l_i], r_i(v) \rangle\right) \in T$ if
   $(l_i, a, g_i, d_i, r_i, l'_i) \in T_i$, $a \in CA$ and $v \models g_i$

2. **(synchronization)** $\left(\langle \bar{l}, v \rangle, a, \left\langle \bar{l}\,[l'_i/l_i][l'_j/l_j], (r_i \cup r_j)(v) \right\rangle\right) \in T$ if
   $(l_i, a!, g_i, d_i, r_i, l'_i) \in T_i$, $(l_j, a?, g_j, d_j, r_j, l'_j) \in T_j$ and $v \models g_i \wedge g_j$ $(i \neq j)$

3. **(delays)** $\left(\langle \bar{l}, v \rangle, \delta, \langle \bar{l}, v + \delta \rangle\right) \in T$ if
   $\delta \in \mathbb{R}^+$ and for all $\delta' \in \mathbb{R}^{+0}$, $\delta' < \delta$: (1) $(v + \delta') \not\models g(t)$ for all $t \in T_i$ $(i : 1..n)$ s.t. $lab(t) \in CA$, $src(t) = l_i$ and $d(t) = eager$; and (2) $(v + \delta') \not\models g(t_i) \wedge g(t_j)$ for all $t_i \in T_i$, $t_j \in T_j$ $(i, j : 1..n, i \neq j)$ s.t. $lab(t_i) = \overline{lab(t_j)}$, $src(t_i) = l_i$, $src(t_j) = l_j$ and $d(t_i) = eager$.

where $\langle \bar{l}, v \rangle \in S$ and $\bar{l} = \langle l_1, \ldots, l_n \rangle$.

A *run* is a finite or countably infinite sequence of transitions in the timed transition system. We say that a run is *infinite* if action transitions occur infinitely often. The *accumulated delay* of a run is the limit of the sum of all delays in the sequence. A run is *time-divergent* if its accumulated delay is infinite. The *projection* of a run ρ over a network component A, is the sequence of actions of A that are embedded in the transitions of ρ.

We say that an action is *enabled* (in a given state) if its source location is in the current location vector, and its guard holds true in the current valuation. We use the term *matching* actions to refer to any pair of half actions, t and $\bar{t}$, s.t. t and $\bar{t}$ are in different components of the network and have complementary labels. We say that an action is *executable* (in a given state) if it is enabled and either is a completed action, or is a half action and there exists an enabled matching action. Matching actions must be executed simultaneously, and half actions cannot be executed autonomously.

DEFINITION **2.1. (Loop)** *Let A be an automaton, and T be the set of all actions in A. A* loop *in A is an elementary cycle in the graph of A, i.e., a sequence $lp = t_0 \ldots t_{n-1}$ where $t_i \in T$, $tgt(t_i) = src(t_{(i+1) \bmod n})$ and $src(t_i) \neq src(t_j)$, for all $0 \leq i \neq j < n$.*

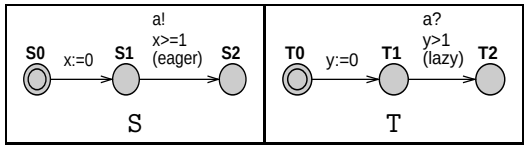

Figure 1: Time-actionlocks in TAD networks

Let lp be a loop in automaton A. Let $l_{0,A}$ denote the initial location of A. Let T_A denote the set of actions in A. $Locs(lp)$ and $Act(S)$ are the sets of all locations and actions in lp, resp. An *entry location* of lp is a location $l \in Locs(lp)$ s.t. $l = l_{0,A}$ or there exists $t \in T_A \setminus Act(lp)$ s.t. $tgt(t) = l$. $EntryLocs(lp)$ is the set of entry locations of lp. $CA(lp)$ and $HA(lp)$ are the sets of completed actions and half actions in lp, resp. We say that lp is a *synchronizing loop* if $HA(lp) \neq \emptyset$, and a *completed loop* otherwise. $Guards(lp)$ and $Resets(lp)$ are the sets of guards and clocks that are reset in lp, resp. Given a set of loops S, we define the generalized sets $Locs(S)$, $Act(S)$, $EntryLocs(S)$, $CA(S)$, $HA(S)$, $Guards(S)$ and $Resets(S)$, as expected.

Definition **2.2. (Traversal)** *Let $|A$ be a TAD network. Let A be a component automaton of $|A$, and let lp be a loop in A. Let ρ be a run in the TAD network. We say that ρ* traverses *lp if the sequence of actions that defines lp is a subsequence in the projection of ρ over A (note that, delays and other components' actions may occur while ρ is traversing lp). Given a set of loops S, a run ρ* traverses *S if ρ traverses each loop in S.*

Definition **2.3. (Zeno run, Time-actionlock and Zeno timelock)** *A* Zeno run *is an infinite run that is not time-divergent (i.e., a run where actions occur infinitely often in finite time). A timelock is a state where time-divergent runs do not exist. We distinguish between* time-actionlocks *and* Zeno timelocks*. A time-actionlock is a timelock where all runs are finite. A Zeno timelock is a timelock, s, where a Zeno run exists from any state that is reachable from s.*

Examples. Figure 1 shows a TAD network with two components, `S` and `T`, where `x` and `y` are local clocks, `a!` is an eager action and `a?` is a lazy action (we have omitted the labels of completed actions, as these are not relevant to the example). Consider any state $s = \langle \bar{l}, v \rangle$, where $\bar{l} = \langle \texttt{S1}, \texttt{T1} \rangle$, $v(\texttt{x}) \geq 1$ and $v(\texttt{y}) \leq 1$. By semantics of eager actions (see the rule for generating delays in the timed transition system), time cannot pass whenever eager actions can be executed, thus the maximum allowed delay from s is $\delta \in \mathbb{R}^{+0}$ s.t. $(v + \delta)(\texttt{y}) = 1$ (note that, any $\delta' > \delta$ would make the eager action `a!` executable). However, neither delays nor actions may occur at $s + \delta$, because `a?` is not yet enabled. By definition, the state s (and any s' reachable from s) is a time-actionlock.

Consider the TAD model shown in fig. 2 (this example is adapted from [3]). The automaton `Lamp` models the behavior of a 3-state lamp. At the pressing of a button (`press?`), the lamp changes from `Off` to `Dim`. If the button is pressed again quickly (in less than 5 t.u.), the state changes to `Bright`. Instead, if the button is pressed after the lamp has been `Dim` for at least 5 t.u., the lamp is switched off. Pressing the button while the lamp is `Bright` also switches it off. Timing constraints are expressed on the clock `x`.

`LazyUser` and `EagerUser` are two different environments for the lamp. The automaton `LazyUser` models a user who can press the button at any time. This is modeled by making `press!` a *lazy* action. The automaton `EagerUser` models a user who continuously attempts to press the button, asap. This is modeled by making `press!` an *eager* action. (The automaton `Clock` will help us illustrate the nature of Zeno behavior).

Consider the network $|A_{\text{lazy}} = \langle \texttt{Lamp}, \texttt{LazyUser}, \texttt{Clock} \rangle$. This network exhibits a Zeno run that cycles through `Off`, `Dim` and `Bright` in less than 5 t.u. (the user can be arbitrarily fast). However,

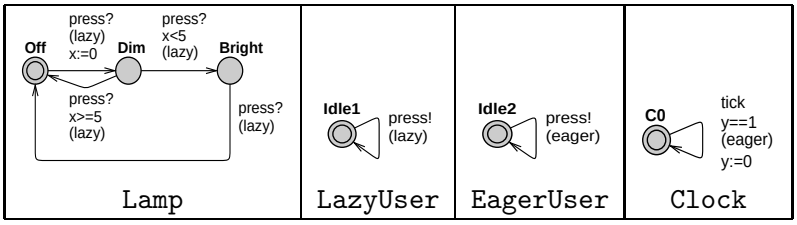

Figure 2: Zeno runs and Zeno-timelocks in TAD

delays are not prevented in any state. Thus, from any state in the behavior of $|A_{\text{lazy}}$, there is at least one infinite, time-divergent run where `Clock` ticks once every time unit. On the other hand, Zeno runs are possible in the network $|A_{\text{eager}} = \langle \texttt{Lamp}, \texttt{EagerUser}, \texttt{Clock} \rangle$, but here the passage of time is prevented in all states. Thus, by definition, all states in the behavior of $|A_{\text{eager}}$ are Zeno-timelocks. Moreover, as components synchronize implicitly on the passage of time, `Clock` never ticks (even though it is conceptually independent from the rest of the network). The global effect of Zeno-timelocks may also be seen in the network $|A_{\text{users}} = \langle \texttt{Lamp}, \texttt{LazyUser}, \texttt{EagerUser} \rangle$, where the behavior of `EagerUser` and `LazyUser` can no longer be distinguished (users' actions may interleave but no delay is allowed between consecutive moves).

In order to see the effect of Zeno behavior on verification, consider the following liveness formulae in TCTL [1].

$$\psi_{\text{weak}} = \forall\Box\exists\Diamond_{\geq 1} true \qquad \psi_{\text{strong}} = \forall\Box\forall\Diamond_{\geq 1} true$$

The formula ψ_{weak} expresses a weak form of time-divergence: ψ_{weak} is satisfiable (in the behavior of a TAD network) if there exists *at least* one time-divergent run at any reachable state. Note that, the satisfiability of ψ_{weak} on the network characterizes the absence of Zeno-timelocks ($|A_{\text{lazy}} \models \psi_{\text{weak}}$ and $|A_{\text{eager}} \not\models \psi_{\text{weak}}$). However, Zeno runs may occur even when ψ_{weak} holds true. Absence of Zeno runs is expressed by the formula ψ_{strong}, which holds true when all infinite runs are time-divergent and all finite runs can be extended to time-divergent runs ($|A_{\text{lazy}} \not\models \psi_{\text{strong}}$ and $|A_{\text{eager}} \not\models \psi_{\text{strong}}$).

2.1 Prevention of Timelocks and Zeno Behavior in TAD networks

Time-reactivity [4] is a desirable property of timed transition systems. This property holds if, from any state, either time may pass or actions can be executed. The following syntactic restriction guarantees the time-reactivity of TAD networks: For any action t, if either $d(t) = \mathit{eager}$ or there exists $\bar{t}$ s.t. $d(\bar{t}) = \mathit{eager}$, then $g(t)$ must be left-closed.[1] This restriction guarantees that delays are prevented only when eager actions can be executed (see the semantic rule for delays). We assume, in this paper, that TAD networks satisfy this requirement. (Note that, action `a?` in fig. 1 is guarded with `y>1`, which is not left-closed.)

On the other hand, Zeno runs and Zeno timelocks are difficult to prevent in TAD networks. Any solution that is implemented at the level of semantics is likely to enforce minimum delays between the consecutive executions of actions, which would severely constrain the set of abstract models that can be written in the formalism.

3 Compositional Detection of Zeno Runs

This section explains how to exploit the syntax of loops in components of a TAD network, in order to discover all possible sources of Zeno runs. This analysis is complemented by TCTL reachability

[1] A guard g is *left-closed* if the interval $\{\delta \in \mathbb{R} \mid (v+\delta) \models g\}$ is either left-closed or left-unbounded, for all $v \in \mathbb{V}$. $v \models g$. For conjunctions of single-clock constraints, g is left-closed if $x > c$ does not occur in g, for any $x \in \mathbb{C}$ and $c \in \mathbb{N}$.

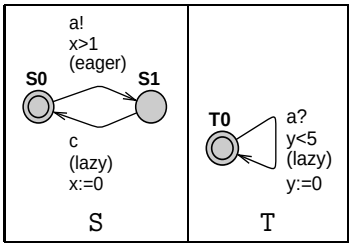

Figure 3: Infinite synchronization between SNZ and NSNZ loops is time-divergent

formulae, which characterize the occurrence of Zeno runs on unsafe loops (i.e., those loops where the absence of Zeno runs could not be guaranteed by the syntactic analysis). The main results are stated in Theorems 3.1 and 3.3.

3.1 A syntactic analysis to confirm the absence of Zeno behavior

We apply Tripakis' *strong non-Zenoness* property [20] to discover those loops which *cannot* be traversed infinitely often by Zeno runs, i.e., those whose syntax force at least one time unit to pass between consecutive iterations.

DEFINITION **3.1. (Strong non-Zenoness)** *A loop is strongly non-Zeno (or SNZ, for short) if there is a clock that is both reset and bounded from below in the loop. Formally, a loop* $lp = t_0 \dots t_{n-1}$ *is a* SNZ loop *if there exists* $x \in \mathbb{C}$, $0 \leq i, j < n$, $m \in \mathbb{N}$, $m > 0$, *s.t.* $x \in r(t_i)$ *and* $g(t_j) \Rightarrow x \geq m$. *(We refer to loops that are not SNZ as* NSNZ loops*.)*

Tripakis proved that Zeno runs cannot occur in timed safety automata [14] if all cycles in the automaton (but not necessarily elementary cycles) are SNZ [20, 19]. In previous work, we exploited synchronization to derive a weaker sufficient condition for the absence of Zeno runs in networks of timed automata [7, 12], which allows NSNZ loops in the network as long as those loops cannot be traversed without simultaneously traversing SNZ loops. This analysis was more efficient and precise than the one originally proposed in [20, 19]; more efficient, because only elementary cycles needed to be considered; more precise, in the sense that it produced fewer false positives, because a bigger class of interactions among loops could be recognized as safe (i.e., time-divergent in the presence of infinite iterations).

The idea in [7, 12] was to identify which groups of NSNZ loops (either completed loops of groups of synchronizing loops with matching actions) could possibly complete an iteration without necessarily synchronizing with SNZ loops. If no such offending group was found, the analysis guaranteed the absence of Zeno runs (and therefore, by definition, the absence of Zeno-timelocks). Otherwise, the occurrence of Zeno runs could not be confirmed. However, the analysis could confirm that, if Zeno runs indeed occurred in the network, these runs must necessarily traverse (infinitely often) all loops in some offending group. Hence, the syntactic analysis identified *all* loops in the network where Zeno runs could possibly occur.

By way of example, Figure 3 shows a network $|A = \langle \mathtt{S}, \mathtt{T} \rangle$ with two synchronizing loops. The loop in `S` is SNZ, because the clock `x` is bounded from below (`x>1`) and reset (`x:=0`) in the loop. The loop in `T` is a NSNZ loop. However, the network is free from Zeno runs, because infinite runs must necessarily traverse the SNZ loop infinitely often, i.e., they have infinite accumulated delay. Equivalently, through synchronization, the SNZ loop prevents the NSNZ loop from being iterated arbitrarily fast.

In what follows, we show that a similar analysis can be applied to TAD networks. Compared to our previous work [7, 12], here we add a simple syntactic check to improve the precision of the analysis. The check aims to eliminate false positives that are produced by NSNZ synchronizing loops that have been

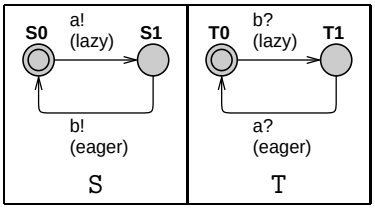

Figure 4: Synchronizing loops with matching actions, which cannot synchronize jointly

grouped together based on their matching actions, but where the ordering of the half actions prevents a complete joint iteration. This class of false positives is illustrated by fig. 4. This figure shows two NSNZ synchronizing loops, one in component `S` and the other in component `T`, which can be grouped together based on matching actions. However, given that the only entry locations are `S0` and `T0`, these loops cannot synchronize jointly to produce complete iterations.

DEFINITION **3.2. (NSNZ group)** *Let $|A$ be a TAD network. A NSNZ group is a finite, non-empty set of loops in the component automata of $|A$, S, s.t. (a) either all loops in S are NSNZ synchronizing loops or S is a singleton which contains a NSNZ completed loop; (b) for any clock $x \in \mathbb{C}$ and action t in some loop in S, if $x \in r(t)$ then there is no action t' in any loop in S s.t. $g(t') \Rightarrow x \geq m$, for any $m \in \mathbb{N}$, $m > 0$; (c) all loops in S, which belong to the same component automaton of $|A$, form a strongly connected subgraph; and (d) $\bar{t} \in HA(S)$ for all $t \in HA(S)$.*

DEFINITION **3.3. (Entry vector)** *Let S be a NSNZ group. Let $A_1, \ldots, A_m$ be all distinct component automata that have at least one loop in S. Let L_i denote the set of locations of A_i, $i : 1..m$. An* entry vector *of S is a location vector of the form $\langle l_1, \ldots, l_m \rangle$, where $l_i \in L_i \cap EntryLocs(S)$ for all $i : 1..m$.*

DEFINITION **3.4. (Unfolding)** *Let S be a NSNZ group. Let ℓ be an entry vector of S. An* unfolding *of S from ℓ is a cycle σ of the form:*

$$\sigma = \ell_1 \xrightarrow{\alpha_1} \ell_2 \ldots \ell_k \xrightarrow{\alpha_k} \ell_1$$

where

1. *$\ell = \ell_1$ and $\ell_i \subseteq Locs(S)$ for all $i : 1..k$;*
2. *$\ell_i \neq \ell_j$ for all $i \neq j$, $i, j : 1..k$;*
3. *either $\ell_{i+1} = \ell_i[tgt(t)/src(t)]$, $\alpha_i = t$, $t \in CA(S)$ and $src(t) \in \ell_i$, or $\ell_{i+1} = \ell_i[tgt(t)/src(t), tgt(\bar{t})/src(\bar{t})]$, $\alpha_i = (t, \bar{t})$, $t, \bar{t} \in HA(S)$ and $src(t), src(\bar{t}) \in \ell_i$, for all $i : 1..k$; and*
4. *for each $t \in Act(S)$, there is some $i : 1..k$ s.t. $\alpha_i = t$ or $\alpha_i = (t, \bar{t})$.*

DEFINITION **3.5. (ZR group)** *A* ZR group *is a NSNZ group, S, s.t. (a) S admits at least one unfolding, and (b) no proper subset $S' \subset S$ admits unfoldings.*

THEOREM **3.1.** *In TAD networks, the absence of ZR groups implies the absence of Zeno behaviors.*

Proof. We prove an equivalent claim: Zeno runs in TAD networks traverse ZR groups infinitely often. Let ρ be a Zeno run. By definition of Zeno runs, ρ must traverse some loop infinitely often (because ρ is an infinite run), but ρ cannot traverse SNZ loops infinitely often (because infinite traversals of SNZ loops entails time-divergence). Then, (infinitely often) either ρ traverses a NSNZ completed loop or a group of NSNZ synchronizing loops. Traversal implies unfolding, hence (by definition) such loops are part of a ZR group. □

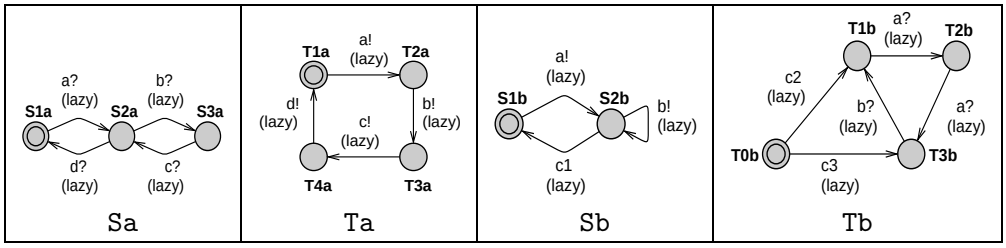

Figure 5: ZR groups

A note on complexity. The number of NSNZ loops in a given component is (at most) exponential in the size of the component (i.e., in the number of locations and transitions in the automaton). However, in practice, the majority of component loops will be SNZ loops, and most of the NSNZ loops will synchronize only with SNZ loops. Hence, the number of NSNZ groups is usually small.

This leaves us with the problem of checking for existence of unfoldings in a given NSNZ group, S. Note that, if S is a single NSNZ completed loop, then S trivially admits unfoldings; else, S is a group of synchronizing NSNZ loops and the interleaving of completed actions in the loops is irrelevant to the existence of unfoldings. Now, given an entry vector ℓ of S, the existence of unfoldings in S from ℓ can be checked with a DFS-based algorithm, which attempts to traverse the loops in S (synchronizing on half actions when necessary) to produce a non-empty sequence of actions that starts and ends in ℓ. Therefore, checking whether S is a ZR group can be solved in space proportional to $|Act(S)|$, and in time proportional to $e \times m^k$, where e is the number of entry vectors of S, m is the maximum number of matching actions $\bar{t}_1, \ldots \bar{t}_m \in HA(S)$ for any $t \in HA(S)$ (where $\bar{t}_i, \bar{t}_j$ are in different loops, for all $1 \leq i \neq i \leq m$), and k is the number of different output actions in $HA(S)$.

Example. Figure 5 shows four components `Sa`, `Ta`, `Sb` and `Tb`. Consider the network $|A = \langle \texttt{Sa}, \texttt{Ta} \rangle$. There are two NSNZ loops in `Sa`, with entry locations `S1a` and `S2a`, and one NSNZ loop in `Ta`, with entry location `T1a`. This is a ZR group with a single unfolding, σ_a.[2]

$$\sigma_a = \langle \texttt{S1a}, \texttt{T1a} \rangle \xrightarrow{(a?,a!)} \xrightarrow{(b?,b!)} \xrightarrow{(c?,c!)} \xrightarrow{(d?,d!)} \langle \texttt{S1a}, \texttt{T1a} \rangle$$

Now, consider the network $|B = \langle \texttt{Sb}, \texttt{Tb} \rangle$. There are two NSNZ synchronizing loops in `Sb`, with entry locations `S1b` and `S2b`, and one NSNZ loop in `Tb`, with two entry locations, `T1b` and `T3b`. There are two possible unfoldings here, σ_b from $\langle \texttt{S1b}, \texttt{T1b} \rangle$ and σ'_b from $\langle \texttt{S2b}, \texttt{T3b} \rangle$, as follows.

$$\sigma_b = \langle \texttt{S1b}, \texttt{T1b} \rangle \xrightarrow{(a?,a!)} \xrightarrow{c1} \xrightarrow{(a?,a!)} \xrightarrow{(b?,b!)} \xrightarrow{c1} \langle \texttt{S1b}, \texttt{T1b} \rangle$$

$$\sigma'_b = \langle \texttt{S2b}, \texttt{T3b} \rangle \xrightarrow{(b?,b!)} \xrightarrow{c1} \xrightarrow{(a?,a!)} \xrightarrow{c1} \xrightarrow{(a?,a!)} \langle \texttt{S2b}, \texttt{T3b} \rangle$$

The previous examples showed that, the existence of unfoldings depend on entry locations, and that a loop may synchronize with many interconnected loops in different components of the network (or with a single loop, more than once) in order to complete an unfolding. The following example shows that several different unfoldings may exist starting in the same entry vector. Figure 6 shows a TAD network with three components `R`, `S`, and `T`. This network exhibits a single ZR group, which is formed by all the component loops and has $\ell = \langle \texttt{R0}, \texttt{S0}, \texttt{T0} \rangle$ as its only entry vector. Two unfoldings are possible from ℓ, σ_R and σ_T, depending on which with component, `R` or `T`, `S` synchronizes first.

[2]When the unfolding in question is clear, we will omit the intermediate location vectors, and we will refer to the actions involved by their labels.

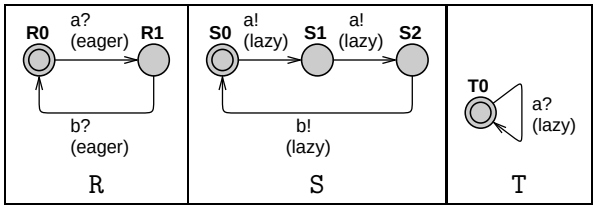

Figure 6: Multiple unfoldings from the same entry vector

$$\sigma_R = \langle \mathtt{R0}, \mathtt{S0}, \mathtt{T0} \rangle \xrightarrow{(a?,a!)} \langle \mathtt{R1}, \mathtt{S1}, \mathtt{T0} \rangle \xrightarrow{(a?,a!)} \langle \mathtt{R1}, \mathtt{S2}, \mathtt{T0} \rangle \xrightarrow{(b?,b!)} \langle \mathtt{R0}, \mathtt{S0}, \mathtt{T0} \rangle$$

$$\sigma_T = \langle \mathtt{R0}, \mathtt{S0}, \mathtt{T0} \rangle \xrightarrow{(a?,a!)} \langle \mathtt{R0}, \mathtt{S1}, \mathtt{T0} \rangle \xrightarrow{(a?,a!)} \langle \mathtt{R1}, \mathtt{S2}, \mathtt{T0} \rangle \xrightarrow{(b?,b!)} \langle \mathtt{R0}, \mathtt{S0}, \mathtt{T0} \rangle$$

3.2 A reachability analysis to confirm the occurrence of Zeno runs

The existence of ZR groups in the network does not imply the occurrence of Zeno runs (i.e., the converse of Theorem 3.1 does not hold). However, for a large class of ZR groups (*convergent ZR groups*), it is possible to obtain a sufficient and necessary condition for the occurrence of Zeno runs, which itself can be expressed by simple reachability formulae in TCTL [14].[3] This is stated in Theorem 3.3.

DEFINITION **3.6. (Convergent ZR group)** *A ZR group S is* convergent *if the conjunct $x > 0$ does not occur in any $g \in Guards(S)$, for any $x \in Resets(S)$.*

LEMMA **3.2.** *Let S be a convergent ZR group, and ρ a Zeno run that traverses S infinitely often. Then, there exists a Zeno run ρ' that traverses S infinitely often, s.t. $delay(\rho') = 0$.*

Proof. Let ρ be a Zeno run that traverses S infinitely often. Let s be any state in ρ that is reachable after ρ has traversed S at least once. W.l.o.g., choose s s.t. $s \xrightarrow{\delta} s' \xrightarrow{a} s''$ are the next two transitions in ρ, where $\delta \in \mathbb{R}^+$ and $a \in CA$. Assume that $t \in Act(S)$ is the completed action in S whose execution is represented by the action transition $s' \xrightarrow{a} s''$ (the argument holds also if $s' \xrightarrow{a} s''$ represents the synchronization of half actions in S). Note that, action t is also enabled in s; otherwise $g(t) \Rightarrow x > 0$ for some clock x which was reset in ρ after the last previous execution of t and before s was reached (which contradicts the premise that S is a convergent ZR group). Then, from s, no delay transition in ρ is necessary for the execution of action transitions (i.e., the valuation at s ensures the traversal of S regardless of delays). Therefore there exists a Zeno run ρ' that traverses S infinitely often, starts in s, s.t. $delay(\rho') = 0$. □

DEFINITION **3.7. (Reachability formula φ_{ZR})** *Let $|A$ be a TAD network. Let S be a ZR group in $|A$. Let $\ell = \langle l_1, \ldots, l_m \rangle$ be an entry vector of S. Let $A(j)$ denote the component in $|A$ that contains l_j, for each $j : 1..m$. We define a TCTL reachability formula, $\varphi_{\mathrm{ZR}}(S, \ell)$, as follows.*

$$\varphi_{\mathrm{ZR}}(S, \ell) \triangleq \exists\Diamond \left(\bigwedge_{j:1..m} A(j).l_j \ \wedge \ \bigwedge_{g \in Guards(S)} g \right)$$

THEOREM **3.3.** *Let $|A$ be a TAD network. Let S be a convergent ZR group in $|A$. The following conditions are equivalent.*

1. *There exists a Zeno run that traverses S infinitely often.*

[3] We consider reachability formulae of the form $\exists\Diamond\phi$, where ϕ is a state formula built from clock constraints, references to automata locations, and Boolean operators.

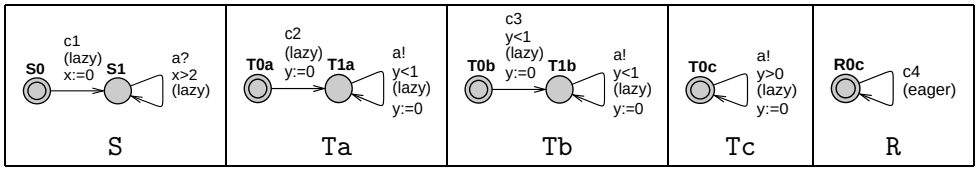

Figure 7: Detecting the occurrence of Zeno runs with reachability formulae

2. $\varphi_{\mathrm{ZR}}(S,\ell)$ holds true in $|A$, for some entry vector ℓ that admits an unfolding of S.

Proof. ((1) $\Rightarrow$ (2)). Let ρ be a Zeno run that traverses S infinitely often. We can choose ρ s.t. it has zero accumulated delay, by Lemma 3.2. Let s be any state in ρ that is reachable after ρ has traversed S at least once, and s.t. the location vector in s contains some entry vector ℓ of S (note that, ℓ admits the unfolding of S that is represented by the next traversal of S by ρ from s). Necessarily, all states in ρ after s (including s) have the same valuation, which simultaneously satisfies all $g \in \mathit{Guards}(S)$. Hence, $s \models \varphi_{\mathrm{ZR}}(S,\ell)$.

((2) $\Rightarrow$ (1)). Let ℓ be an entry vector which admits an unfolding of S. Let $s = \langle \bar{l}, v\rangle \models \varphi_{\mathrm{ZR}}(S,\ell)$, i.e., $\bar{l}$ includes ℓ and v satisfies all $g \in \mathit{Guards}(S)$. Necessarily, a state s_0 can be reached from s by traversing S without delay. (Otherwise, given that ℓ admits an unfolding of S and v satisfies all $g \in \mathit{Guards}(S)$, it must be the case that $g \Rightarrow x > 0$ for some $x \in \mathit{Resets}(S)$ and $g \in \mathit{Guards}(S)$, which contradicts the premise that S is a convergent ZR group.) Then, there must exist an elementary cycle in the timed transition system of $|A$, of the form: $s_0 \xrightarrow{a_0} s_1 \ldots s_{n-1} \xrightarrow{a_{n-1}} s_n$, where $s_0 = s_n$, each transition $s_i \xrightarrow{a_i} s_{i+1}$ represents the execution of some action in S, and $v_0 = \ldots = v_n = \mathit{Resets}(S)(v)$. This cycle represents a Zeno run. □

Example. Figure 7 shows five components, `S`, `Ta`, `Tb`, `Tc` and `R`. Consider the TAD network $|A = \langle \mathtt{S}, \mathtt{Ta}\rangle$, the ZR group S_a defined by the loops at `S1` and `T1a`, and the entry vector $\ell_a = \langle \mathtt{S1}, \mathtt{T1a}\rangle$. For S_a to exhibit Zeno runs, $\ell_a = \langle \mathtt{S1}, \mathtt{T1a}\rangle$ should be reachable with a valuation v that satisfies $v(\mathtt{x}) > 2$ and $v(\mathtt{y}) < 1$. This state witnesses $\varphi_{\mathrm{ZR}}(S_a, \ell_a)$, which is defined as follows.

$$\varphi_{\mathrm{ZR}}(S_a,\ell_a) \quad = \exists\Diamond(\mathtt{S.S1} \wedge \mathtt{Ta.T1a}) \wedge \mathtt{x} > 2 \wedge \mathtt{y} < 1)$$

On the other hand, consider the network $|B = \langle \mathtt{S}, \mathtt{Tb}\rangle$, the ZR group S_b defined by the loops at `S1` and `T1b`, and the entry vector $\ell_b = \langle \mathtt{S1}, \mathtt{T1b}\rangle$. Zeno runs cannot occur in S_b, because ℓ_b cannot be reached with a valuation v that satisfies $v(\mathtt{x}) > 2$ and $v(\mathtt{y}) < 1$ (this is prevented by the guard `y<1` in completed action `c3`). Correspondingly, the formula $\varphi_{\mathrm{ZR}}(S_b, \ell_b)$ fails to hold in $|B$.

$$\varphi_{\mathrm{ZR}}(S_b,\ell_b) \quad = \exists\Diamond(\mathtt{S.S1} \wedge \mathtt{Tb.T1b} \wedge \mathtt{x} > 2 \wedge \mathtt{y} < 1)$$

Finally, let us show an example that justifies the restriction imposed by convergent ZR groups. consider the network $|C = \langle \mathtt{S}, \mathtt{Tc}, \mathtt{R}\rangle$. This network contains a non-convergent ZR group, S_c, defined by the loops at `S1` and `T1c` (note that, `y>0` and `y:=0` occur in the self loop at `T1c`). S_c has a single entry vector, $\ell_c = \langle \mathtt{S1}, \mathtt{T1b}\rangle$. The formula $\varphi_{\mathrm{ZR}}(S_c, \ell_c)$ holds true in $|A$; however, Zeno runs do not occur on S_c because delays are prevented by the eager loop in `R0`. Thus, in general, φ_{ZR} cannot be applied over non-convergent ZR groups.

4 Compositional detection of Zeno-timelocks

This section presents a syntactic and compositional analysis to identify all sources of Zeno-timelocks in a TAD network, based on the syntax of ZR groups. This is complemented by TCTL reachability

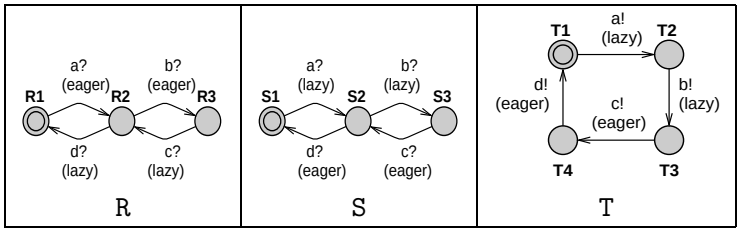

Figure 8: ZT groups

formulae, which characterize the occurrence of Zeno-timelocks that are local to the ZR group (i.e., Zeno-timelocks whose witness Zeno runs are confined to the loops in the ZR group). The main results are stated in Theorems 4.1 and 4.2.

4.1 A syntactic analysis to confirm the absence of Zeno-timelocks

We prove that, the occurrence of Zeno-timelocks in TAD networks imply the existence of ZR groups, which admit unfoldings where eager actions are executed at every step. (We refer to such groups as *ZT groups*.) This yields a syntactic analysis that guarantees the absence (but not the occurrence) of Zeno-timelocks in a TAD network.

DEFINITION **4.1. (ZT group)** *Let $|A$ be a TAD network. Let S be a ZR group in $|A$. We say that S is a* ZT group *if S admits an unfolding, σ, s.t. σ is of the form:*

$$\sigma = \ell_1 \xrightarrow{\alpha_1} \ell_2 \ldots \ell_k \xrightarrow{\alpha_k} \ell_1$$

where, for all $i : 1..k$, there is some eager action $t \in Act(S)$ s.t. $\alpha_i = t$ or $\alpha_i = (t, \bar{t})$. We will refer to σ as an eager unfolding *of S from ℓ_1.*

THEOREM **4.1.** *In TAD networks, the absence of ZT groups implies the absence of Zeno-timelocks.*

Proof. We prove an equivalent claim: The occurrence of Zeno-timelocks in TAD networks imply the existence of ZT groups. Let s be a Zeno-timelock in the TAD network. By definition of Zeno-timelocks, (a) there are no time-divergent runs from s, and (b) there are Zeno runs starting in any state that is reachable from s. From (a), it follows that, from any s' that is reachable from s, there exists some eager action t in the TAD network, and some $\delta \in \mathbb{R}^{+0}$, s.t. t is executable in $s' + \delta$. From (b) and the latter observation, there exists a set of eager actions in the TAD network, E say, which may be executed infinitely often from s. By definition, there exists a ZT group S s.t. $Act(S) \subseteq E$. □

Example. Figure 8 shows three components, `R`, `S` and `T`. Two ZR groups can be formed, S_{RT}, with the loops in `R` and `T`, and S_{ST}, with the loops in `S` and `T`. Note that, only S_{RT} is a ZT group, which admits the eager unfolding σ_{RT}. Instead, the only unfolding of S_{ST} contains moves that do not involve eager actions.

$$\sigma_{RT} = \langle \texttt{R1}, \texttt{T1} \rangle \xrightarrow{a?||a!} \xrightarrow{b?||b!} \xrightarrow{c?||c!} \xrightarrow{d?||d!} \langle \texttt{R1}, \texttt{T1} \rangle$$

Note that, once a Zeno-timelock s is reached, it is possible that some of the Zeno runs starting in s do not execute eager actions. Nonetheless, Theorem 4.1 guarantees the existence of Zeno runs that traverse at least one ZT group infinitely often. Thus, we focus our syntactic analysis (and the reachability analysis discussed in the next section) on the construction of ZT groups.

4.2 A reachability analysis to confirm the occurrence of Zeno-timelocks

The existence of ZT groups does not imply the occurrence of Zeno-timelocks (the converse of Theorem 4.1 does not hold). There are several reasons why a given ZT group, S say, may not induce a Zeno-timelock. Clearly, as for ZR groups, S may not be reachable with valuations that permit Zeno runs. A less obvious reason is that S may contain non-eager unfoldings, which may be executed by time-divergent runs. In addition, even if S is traversed by Zeno runs and it only admits eager unfoldings, time-divergent runs may exist which start in some location of S but are not confined to the loops in S, thus escaping the urgency constraints. Unfortunately, whether Zeno runs are confined to S cannot be characterized as a reachability property.

Nonetheless, reachability analysis suffices to confirm that a given ZT group, S, can be reached with a valuation, v, s.t. (a) v allows S to be traversed by Zeno runs with zero accumulated delay, and (b) v simultaneously disables all actions t s.t. $src(t) \in Locs(S)$ and $tgt(t) \notin Locs(S)$. Thus, once S is reached with v, all runs thereafter are confined to S. We will derive a TCTL reachability property from the syntax of ZT groups, for those ZT groups that only admit eager unfoldings. This property guarantees the occurrence of Zeno-timelocks in the group when it holds true (however, in general, nothing can be concluded when the property fails to hold).

DEFINITION **4.2. (Escape action)** *Let $|A$ be a TAD network. Let T be the set of all actions in $|A$. Let S be a ZT group in $|A$. An* escape action *of S is an action $t \in T$ s.t. $src(t) \in Loc(S)$ and $tgt(t) \notin Loc(S)$.*

Let S be a ZT group. Let $EscAct(S) \subseteq Act(S)$ be the set of escape actions in S. Let $EscCA(S), EscHA(S) \subseteq EscAct(S)$ be the subsets of escape completed actions and escape eager actions in S, respectively.

DEFINITION **4.3. (Local Zeno-timelock)** *Let S be a ZT group. A Zeno-timelock, s, is* local *in S if (a) a Zeno run exists that starts in s and traverses S infinitely often, and (b) no escape action $t \in EscAct(S)$ is executable in any state s' that is reachable from s.*

DEFINITION **4.4. (Reachability formula φ_{ZT})** *Let $|A$ be a TAD network, and T be the set of actions in $|A$. Let S be a ZT group of $|A$. Let $\ell = \langle l_1, \ldots, l_m \rangle$ be an entry vector of S. Let $A(j)$ denote the component in $|A$ that contains l_j, for each $j : 1..m$. We define the reachability formula, $\varphi_{\mathrm{ZT}}(S, \ell)$, as follows.*

$$\varphi_{\mathrm{ZT}}(S,\ell) = \exists\Diamond\,(\ \ \bigwedge_{j:1..m} A(j).l_j \ \wedge\ \bigwedge_{g\in Guards(S)} g \wedge \bigwedge_{t\in EscCA(S)} \neg\, g(t) \ \wedge\ \bigwedge_{t\in EscHA(S),\bar{t}\in T} \neg\,(g(t)\wedge g(\bar{t})))$$

THEOREM **4.2.** *Let S be a ZT group. Let ℓ be an entry vector of S that only admits eager unfoldings. If $\varphi_{\mathrm{ZT}}(S, \ell)$ holds true in $|A$, then a Zeno-timelock occurs that is local in S.*

Proof. Let $s = \langle \bar{l}, v \rangle$ be any state s.t. $s \models \varphi_{\mathrm{ZT}}(S, \ell)$. Note that, (a) $\varphi_{\mathrm{ZT}}(S, \ell)$ implies $\varphi_{\mathrm{ZR}}(S, \ell)$, and (b) S is a convergent ZR group. (Suppose S is not a convergent ZR group. By definition, there exists some $t \in Act(S)$ and $x \in \mathbb{C}$ s.t. $x > 0$ is a conjunct in $g(t)$. Also, either t or $\bar{t} \in Act(S)$ is an eager action, because only eager unfoldings exist from ℓ. But t is not left-closed, which contradicts our well-formedness assumption on TAD networks § 2). Thus, by Theorem 3.3, there exists a Zeno run, ρ, which starts in s, traverses S infinitely often, s.t. $delay(\rho) = 0$. Now, consider any state $s' = \langle l', v' \rangle$ that is reachable from s. The projection of $\bar{l}'$ over the component automata of S is a location vector in some eager unfolding of S from ℓ, because v disables all escape actions from S. In addition, time cannot pass from s', because the existence of eager unfoldings (and the fact that v satisfies all clock constraints of actions in S) ensures that eager actions exist that are executable in s'. Hence, by definition, s is a Zeno-timelock that is local to S. □

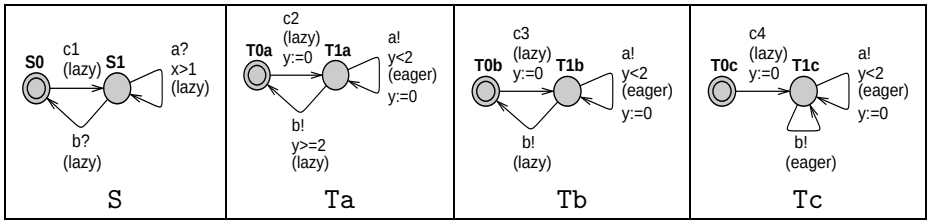

Figure 9: Detecting the occurrence of Zeno-timelocks with reachability formulae

Example. Figure 9 shows four components, `S`, `Ta`, `Tb` and `Tc`. Consider the TAD network $|A = \langle \texttt{S}, \texttt{Ta} \rangle$ and the ZT group S_a, formed by the loops synchronizing on `a!` and `a?`. The entry vector is $\ell_a = \langle \texttt{S1}, \texttt{T1a} \rangle$. Zeno-timelocks occur in S_a with valuation v s.t. $v(\texttt{x}) > 1$ and $v(\texttt{y}) = 0$ (Zeno runs occur that disable the escape actions `b?` and `b!`). This is confirmed by the reachability analysis, as $\varphi_{\text{ZT}}(S_a, \ell_a)$ holds true in $|A$.

$$\varphi_{\text{ZT}}(S_a, \ell_a) \quad = \exists\Diamond(\texttt{S.S1} \wedge \texttt{Ta.T1a} \wedge \texttt{x} > 1 \wedge \texttt{y} < 2 \wedge \neg\,(\mathit{true} \wedge \texttt{y} \geq 2))$$

In contrast, consider the TAD network $|B = \langle \texttt{S}, \texttt{Tb} \rangle$ and the ZT group S_b, formed by the loops synchronizing on `a!` and `a?`. The entry vector is $\ell_b = \langle \texttt{S1}, \texttt{T1b} \rangle$. Zeno-timelocks cannot occur in S_b, because any valuation that reaches ℓ_b and allows the traversal of S_b, also enables the escape actions `b!` and `b?` (the synchronization `b!||b?` leads back to $\langle \texttt{S0}, \texttt{T0b} \rangle$, where time may pass again). The reachability formula, $\varphi_{\text{ZT}}(S_b, \ell_b)$, fails to hold in $|B$ (it is, in fact, trivially unsatisfiable).

$$\varphi_{\text{ZT}}(S_b, \ell_b) \quad = \exists\Diamond(\texttt{S.S1} \wedge \texttt{Tb.T1b} \wedge \texttt{x} > 1 \wedge \texttt{y} < 2 \wedge \neg\,(\mathit{true} \wedge \mathit{true})) = \mathit{false}$$

The previous example showed a case where the characteristic reachability formula failed to hold, and no Zeno-timelock occurred in the network. The following example shows that, in general, nothing can be inferred when the reachability formula fails to hold. Consider the TAD network $|C = \langle \texttt{S}, \texttt{Tc} \rangle$ and the ZT group S_c, formed by the loops synchronizing on `a!` and `a?`. The entry vector is $\ell_c = \langle \texttt{S1}, \texttt{T1c} \rangle$. Here, a Zeno-timelock occurs that is not local to S_c, and this cannot be inferred by reachability analysis. Note that, when ℓ_c is reached with a valuation v, s.t. $v(\texttt{x}) > 1$ and $v(\texttt{y}) = 0$, Zeno runs exist that traverse S_c but also the loops on `b!||b?`, and the passage of time is prevented in all states because `b!` is eager (here, `b?` is the only escape action of S_c). However, the reachability formula, $\varphi_{\text{ZT}}(S_c, \ell_c)$, fails to hold in $|C$ because any valuation that allows the traversal of S_c also enables synchronization on `b!||b?`.

$$\varphi_{\text{ZT}}(S_c, \ell_c) \quad = \exists\Diamond(\texttt{S.S1} \wedge \texttt{Tc.T1c} \wedge \texttt{x} > 1 \wedge \texttt{y} < 2 \wedge \neg\,(\mathit{true} \wedge \mathit{true})) = \mathit{false}$$

5 Conclusions and Future Work

We proposed a number of methods to check for Zeno behavior in TAD networks. These methods are based on a compositional application of Tripakis' strong non-Zenoness property, which works on the syntax of loops (elementary cycles) in component automata. These methods are compositional, in the sense that they are informed by the syntax of loops in component automata and do not require the construction of the network's product automaton.

A purely syntactic analysis was proposed to confirm the absence of Zeno runs, which either guarantees that the TAD network is free from Zeno runs, or it identifies all groups of loops that can potentially be traversed by Zeno runs, infinitely often. This method is complemented by TCTL reachability formulae, which are derived from the syntax of offending loops (i.e., those groups of loops identified by the

syntactic analysis), and which hold true when Zeno runs occur in the loops. Thus, reachability analysis offers a way to reduce false positives. These methods were extended to check for Zeno-timelocks, based on the identification of groups of NSNZ loops, which both may be traversed by Zeno runs and iterate using eager actions at every step. However, reachability analysis could only characterize the occurrence of certain classes of Zeno-timelocks, namely, those whose Zeno runs are confined to a given set of loops.

We would argue that, compositionality was facilitated by the semantics of deadlines. Although, it is possible to port these methods to TA networks, such analysis would be more difficult to implement because invariants express urgency indirectly.

Future work will consider the implementation of these methods, and the integration with available tools for the analysis of TAD models. We believe that, this combined approach is more efficient than verification of time divergence by liveness properties. Moreover, the proposed TCTL formulae can be verified with the simplest form of reachability analysis supported in verification tools (for instance, these formulae can be verified in the IF toolset; and also in Uppaal, given the translation suggested in [10, 9]). Our syntactic analysis can also be applied to TAD models with discrete variables on finite domains (i.e., similar to the extensions of timed automata supported by Uppaal). On the other hand, due to interactions between data variables, the reachability formulae no longer characterize Zeno runs. For instance, these formulae may be satisfiable by valuations that allow only finitely many consecutive iterations (which prevents the occurrence of Zeno runs). Some form of termination analysis may help to identify such cases, but this is subject of further research.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

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

[3] G. Behrmann, A. David, and K. Larsen. A tutorial on Uppaal. In *SFM-RT 2004*, LNCS 3185, pages 200–236. Springer, 2004.

[4] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS 1386, pages 49–63. Springer, 1998.

[5] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Proc. of COMPOS 1997*, LNCS 1536, pages 103–129. Springer, 1998.

[6] H. Bowman. Time and action lock freedom properties for timed automata. In *Proceedings of FORTE 2001*, pages 119–134. Kluwer Academic, 2001.

[7] H. Bowman and R. Gomez. How to stop time stopping. *Formal Aspects of Computing*, 18(4):459–493, December 2006.

[8] M. Bozga, S. Graf, Ileana Ober, Iulian Ober, and J. Sifakis. The IF toolset. In *SFM-RT 2004*, LNCS 3185, pages 237–267. Springer, 2004.

[9] R. Gomez. Verification of Timed Automata with Deadlines in Uppaal. TR 2-08-2008, Computing Laboratory, University of Kent, 2008.

[10] R. Gomez. A compositional translation from Timed Automata with Deadlines to Uppaal Timed Automata. In *7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, Budapest, Hungary, September 2009. Springer. (To appear).

[11] R. Gomez and H. Bowman. Efficient Detection of Zeno Runs in Timed Automata. In J.-F. Raskin and P.S. Thiagarajan, editors, *5th International Conference FORMATS 2007 (Formal Modelling and Analysis of Timed Systems)*, volume 4763 of *LNCS*, pages 195–210, Salzburg, Austria, October 2007. Springer.

[12] R. Gomez and H. Bowman. Efficient Detection of Zeno Runs in Timed Automata. In *5th International Conference on Formal Modelling and Analysis of Timed Systems, FORMATS 2007*, LNCS 4763, pages 195–210. Springer, 2007.

[13] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2):273–337, 1994.

[14] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[15] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proc. of CONCUR 1990*, pages 401–415. Springer-Verlag New York, Inc., 1990.

[16] T. Regan. Multimedia in temporal LOTOS: A lip synchronisation algorithm. In *PSTV XIII, 13th Protocol Spec., Testing & Verification*. North-Holland, 1993.

[17] S. Schneider. *Concurrent and Real-time Systems, the CSP Approach*. Wiley, 2000.

[18] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proc. of STACS'96*, LNCS 1046, pages 347–359. Springer-Verlag, 1996.

[19] S. Tripakis. *The analysis of timed systems in practice*. PhD thesis, Universite Joseph Fourier, Grenoble, France, December 1998.

[20] S. Tripakis. Verifying progress in timed systems. In *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop*, LNCS 1601. Springer-Verlag, 1999.

[21] F. Wang. Model-checking distributed real-time systems with states, events, and multiple fairness assumptions. In *Proceedings of AMAST 2004*, volume 3116 of *LNCS*, pages 553–568. Springer, 2004.

[22] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1-2):123–133, 1997.