Model Checking MITL formulae on Timed Automata: a Logic-Based Approach

CLAUDIO MENGHI, SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
MARCELLO M. BERSANI, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
MATTEO ROSSI, Dipartimento di Meccanica, Politecnico di Milano, Italy
PIERLUIGI SAN PIETRO, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

Timed Automata (TA) is de facto a standard modelling formalism to represent systems when the interest is the analysis of their behaviour as time progresses. This modelling formalism is mostly used for checking whether the behaviours of a system satisfy a set of properties of interest. Even if efficient model-checkers for Timed Automata exist, these tools are not easily configurable. First, they are not designed to easily allow adding new Timed Automata constructs, such as new synchronization mechanisms or communication procedures, but they assume a fixed set of Timed Automata constructs. Second, they usually do not support the Metric Interval Temporal Logic (MITL) and rely on a precise semantics for the logic in which the property of interest is specified which cannot be easily modified and customized. Finally, they do not easily allow using different solvers that may speed up verification in different contexts.

This paper presents a novel technique to perform model checking of Metric Interval Temporal Logic (MITL) properties on TA. The technique relies on the translation of both the TA and the MITL formula into an intermediate Constraint LTL over clocks (CLTLoc) formula which is verified through an available decision procedure. The technique is flexible since the intermediate logic allows the encoding of new semantics as well as new TA constructs, by just adding new CLTLoc formulae. Furthermore, our technique is not bound to a specific solver as the intermediate CLTLoc formula can be verified using different procedures.

CCS Concepts: • **Theory of computation → Verification by model checking**; **Logic and verification**; **Logic**;

Additional Key Words and Phrases: Model Checking, Timed Automaton, Signal-Based Semantics

ACM Reference Format:
Claudio Menghi, Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. 2020. Model Checking MITL formulae on Timed Automata: a Logic-Based Approach. 1, 1 (February 2020), 44 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Claudio Menghi, SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, H29 Avenue John F. Kennedy, Luxembourg, 1855, Luxembourg, claudio.menghi@uni.lu; Marcello M. Bersani, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, via Golgi 42, Milan, 20133, Italy, marcello.bersani@polimi.it; Matteo Rossi, Dipartimento di Meccanica, Politecnico di Milano, via Golgi 42, Milan, 20133, Italy, matteo.rossi@polimi.it; Pierluigi San Pietro, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, via Golgi 42, Milan, 20133, Italy, pierluigi.sanpietro@polimi.it.

1 INTRODUCTION

Model checking is an automatic technique to verify whether a model of a system satisfies a property of interest. Various formalisms have been proposed for representing the model and its properties, often in terms of state machines and temporal logics, both having specific peculiarities depending on the designer's goals and tool availability.

Timed Automata [4] (TA) are one of the most popular formalisms to describe system behavior when real time constraints are important. Various tools are available to verify TA: Kronos [33], the de facto standard tool Uppaal [26], RED [32], $\text{Mitl}_{0,\infty}$BMC tool [25] and MCMT [16] (though the latter can only perform reachability analysis of—parametric—networks of TA).

We believe that novel model checking tools for TA should address three main challenges: (C1) providing different semantics of TA including the *continuous time*; (C2) supporting high level expressive complex logics that easily allow the specification of the properties of interest, such as the Metric Interval Temporal Logic (MITL); and (C3) being extensible, i.e., allowing users to add new constructs easily, such as adding new synchronization mechanisms or communication procedures for TA. The main issues related to those challenges are summarized hereafter.

C1. The paradigm of time that is overwhelmingly adopted in practice is based on timed words [4]—i.e., infinite words where each symbol is associated with a real-valued time-stamp—and most tools (except [25]) are founded on such semantics. The so-called *signal-based semantics* is a different interpretation, where each instant of a dense temporal domain (e.g., $\mathbb{R}_{\geq 0}$) is associated with a state, called a signal. Signals are more expressive than timed words (as proved in [20]), thus allowing a more precise representation of the system state over time. In particular, if a signal changes its value at an instant t, it is possible to specify the value of the signal both in t (the "signal edge") and in arbitrary small neighborhoods of t. This allows, for instance, to represent the location of an automaton both just before and immediately after an instantaneous state transition. Despite its greater expressiveness, signal-based semantics has been so far confined mainly to theoretical investigations [5, 12, 20, 31] and seldom used in practice [25], due the difficulty in developing a feasible decision procedure. More precisely, Kindermann et al. [25] implemented a decision procedure for BMC of TA against $\text{MITL}_{0,\infty}$ which is based on the so-called "super-dense" time (also adopted by Uppaal). Under a super-dense time assumption, a TA can fire more than one transition in the same (absolute) time instant; thus, two or more transitions can be fired one after the other and produce many simultaneous, but distinct, configuration changes such that time does not progress. Super-dense time is a modeling abstraction to represent systems that are much faster than the environment they operate in, so their reaction to external events has a negligible delay. In the current work, the signal-based semantics is not "super-dense", i.e., at any time instant each TA is in exactly one state. This choice is mainly dictated by the use of [9] to translate MITL to CLTLoc, which is defined over the a more "traditional" dense-time; still, CLTLoc may be extended to super-dense time.

C2. Temporal Logics with a metric on time, such as MITL [5], have been proposed to specify real-time properties, but they are not fully supported by TA verification tools, that typically provide just some baseline functionalities to address reachability problems (safety assessment) or to perform model-checking of temporal logics without metric (e.g., LTL, CTL) or of fragments of Timed CTL [33]. For example, Uppaal [26] supports only a limited set of reachability properties. However, its ability to express that a certain condition triggers a reaction within a certain amount of time provides a clear improvement over being able only to specify that a reaction will eventually occur. The gap of almost 20 years between the proof of the decidability of MITL [5] and its applicability in practice can be justified mainly with the practical complexity of the underlying decision procedure, hampering the development of efficient tools, until more recent developments of new decision procedures, typically based on efficient SMT-solvers. In fact, both [8, 25] developed a decision procedure for the satisfiability of MITL, a problem which very recently was also tackled by [15]. In particular, [25] proposed a verification procedure for a fragment of MITL, namely $\mathrm{MITL}_{0,\infty}$, on TA, but under a semantics based on super-dense signals. It is however still unknown whether $\mathrm{MITL}_{0,\infty}$ is equivalent to MITL under the latter semantics. There are several procedures to convert MITL specifications into TA [5, 28], but, to the best of our knowledge, only one has been successfully implemented [13–15]. These procedures could be the basis for a model checking tool that (i) transforms the MITL formula to be checked into (a set of) TA; (ii) combines the obtained automata with a network of TA modeling the system; and (iii) performs an emptiness check on the result. However, there is no available automated tool supporting this complete workflow, even in the case of a system modeled by a single TA, as the approach still poses significant conceptual and technical problems—for example when the system is modeled through a network of interacting automata (see Section 6 for further details). In this work, we follow a different approach, entirely based on temporal logic, which allowed us to overcome these problems.

To the best of the authors' knowledge, a verification procedure supporting MITL over (standard) signals is still not available. A proof of the language equivalence of TA and CLTLoc over timed words was given for the first time in [10]; however, the translation presented therein did not consider signals and had only the purpose of proving the equivalence of the formalisms, rather than being intended to be implemented in a tool. For instance, it makes use of many additional clocks that would hinder the performance of any decision procedure. Those limitations fostered the definition of a new, more practical translation, which is also radically different. The new encoding has been devised to be as optimized and extensible as possible, rather than being intended to prove language-theoretical results. Moreover, it also supports networks of TA, whose traces are interpreted for the evaluation of MITL formulae over atomic propositions and arithmetical formulae of the form $n \sim d$, where n is an integer variable manipulated by the automata; to this end, the new encoding allows the representation of the signals associated with the atomic propositions on the locations and with the integer variables elaborated by the network. The new encoding also includes three synchronization primitives and allows the representation of the signal edges at the instant where transitions are taken.

C3. Even if a variety of tools supporting the analysis of TA and networks of TA is available, they usually are not easy to tailor and extend. (1) They only provide a fixed set of modeling constructs that support designers in modeling the system under development, but which are not easily modifiable and customizable. Typical examples are discrete variables (often on finite domains) as well as some communication and synchronization features among different TA. For example, Uppaal provides designers with binary and broadcast synchronization primitives, whereas RED offers sending/receiving communication features via finite FIFO channels. However, often new modeling requirements may prompt designers to formulate specific communication/synchronization features, also based on data structures such as queues, stacks, priority mechanisms. Common model checkers do not explicitly support extending their features and constructs in the above directions, since this could cause a significant variation of the underlying semantics. Ad hoc modifications of a tool are often possible, but they may require a deep knowledge of the tool internals, whose software implementation may be quite complex (depending also in the architecture and the programming language) and scantly documented. (2) The existing model checkers are typically solver-dependent, since they rely on a strong relation between the problem domain and the solution domain—i.e., respectively, the models to be verified and the input language that is used by a verification engine. (3) The cited tools explicitly support only TA, but not other timed formalisms such as, for instance, Time Petri Nets [21, 30], unless an ad hoc front-end is developed (as for instance done by the Romeo

$$\begin{tikzcd} \text{TA} \arrow[dr, "\text{Sect.4}"'] & & \text{MITL } \phi \arrow[dl, "{[9]}"] \\ & \text{CLTLoc} \arrow[d, "{[11]}"] & \\ & \text{CLTLoc solver} & \end{tikzcd}$$

Fig. 1. A generic framework for checking the satisfaction of MITL formulae on TA.

toolkit [27]). Hence, they not easily allow multi-formalism analysis [7], e.g., to design systems with heterogeneous components which are more naturally modeled by different formalisms.

Contribution. This paper describes a novel technique to model check networks of TA over properties expressed in MITL over signals, by relying on a purely logic-based approach. The technique is exemplified in the diagram of Fig. 1. It is based on the solution presented in [9] to translate both a MITL formula and a TA into an intermediate logical language, which is then encoded into the language of the underlying solver. This intermediate level has thus a similar role as the Java bytecode for Java program execution on different architectures. The advantages are that, on the one hand, new TA constructs, logic formalisms or semantics can be dealt with by defining new encodings into the intermediate language; on the other hand, the intermediate language can independently be "ported" to different (possibly more efficient) solvers, by translating into the respective solver languages.

A TA and a MITL formula are translated into CLTLoc, a metric temporal logic [11]. CLTLoc is a decidable extension of Linear Temporal Logic (LTL) including real-valued variables that behave like TA clocks. The satisfiability of CLTLoc can be checked by using different procedures; a bounded approach based on SMT-solvers is available as part of the Zot formal verification tool [6]. This intermediate language easily allows for different semantics of TA such as, for instance, the signal edges that are generated by the TA when transitions are fired (see Sect. 3). Moreover, different features of the TA modeling language can be introduced by simply adding or changing formulae in the CLTLoc encoding. As an example, finite queues or other data structures can be easily included as long as the new features can be expressed in terms of CLTLoc formulae. The CLTLoc formula encoding the network of TA and the MITL property is modular, in the sense that the parts that translate the MITL property are separated from the formulae translating the TA network. Moreover, each aspect of the semantics of the (network of) TA is isolated in a specific formula, with only few interconnecting points with the other parts. Therefore, each part of the resulting translation is self-reliant, thus easily allowing changes or extensions. In this work we consider three different baseline semantics for the shape of the edge with which signals change values, which correspond to different types of signals that can be generated by a (network) of TA (Table 7). Furthermore, we formalize three different baseline semantics that can be introduced depending on the synchronization constructs (Tables 2), and four different baseline semantics that can be defined for the liveness of the transitions (Tables 1). For each of these semantics an encoding into intermediate CLTLoc formulae is presented. The semantics of the (network of) TA, and the intermediate CLTLoc encoding, depends on the baseline semantics chosen for the constructs of the TA.

The technique presented in this work is implemented in a Java tool, called TACK (https://github.com/claudiomenghi/TACK), which is built on the QTLSolver (https://github.com/fm-polimi/

qtlsolver), and extends the translation of [11] to deal with a network of TA and to add a new front-end for the specification of the network and its properties. TACK takes as input a (network of) TA, described with a syntax compatible with Uppaal, and the MITL property to be verified. Unlike Uppaal, TA and MITL are interpreted according to the *signal-based semantics*. The CLTLoc formula produced by TACK is then fed to Zot for automated verification.

To evaluate the benefits that ensue from the adoption of an intermediate language, this work shows how to deal with different signal-based semantics, synchronization primitives and liveness conditions. Furthermore, to show the flexibility achieved by decoupling the model-checking problem and the resolution technique, different solvers are employed for verifying the intermediate CLTLoc encoding. The efficiency of technique is evaluated over some standard benchmarks, namely the Fischer (see e.g. [3]), the CSMA/CD (see e.g. [1]) and the Token Ring (see e.g. [23]) protocols. The Fischer protocol was verified also through the $\text{MITL}_{0,\infty}$BMC tool for a partial comparison (where possible and reasonably meaningful) of the two approaches. We also study the timed lamp model verified in [15] for a qualitative comparison with the approach introduced in that work.

The paper is structured as follows. Section 2 presents the background and the notation used in the rest of this work. Section 3 introduces the continuous time semantics of TA. Section 4 presents the algorithm to convert a TA into a CLTLoc formula. Section 5 describes the model checking algorithm to verify MITL formulae on automata with time. Section 6 evaluates TACK and discusses the experimental results. Section 7 concludes.

2 BACKGROUND

This section presents TA (enriched with integer-valued variables and synchronization), MITL and CLTLoc.

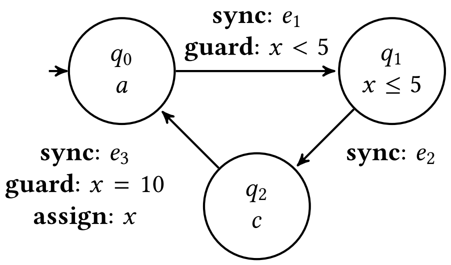
2.1 Timed automata

Let X be a finite set of *clocks* with values in $\mathbb{R}$. $\Gamma(X)$ is the set of *clock constraints* over X defined by the syntax $\gamma := x \sim c \mid \neg\gamma \mid \gamma \wedge \gamma$, where $\sim \in \{<, =\}$, $x \in X$ and $c \in \mathbb{N}$. Let Act be a set of events, Act_τ is the set $Act \cup \{\tau\}$, where τ is used to indicate a null event. Finally, we indicate by $\wp$ the power set operator.

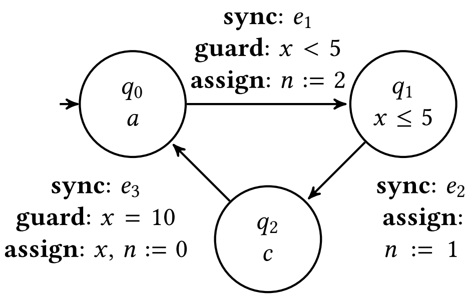
Definition 2.1 (Timed Automaton). Let AP be a non-empty set of atomic propositions, X be a set of clocks and Act be a set of events. A *Timed Automaton* is a tuple $\mathcal{A} = \langle AP, X, Act_\tau, Q, q_0, Inv, L, T\rangle$, where: Q is a finite set of control states (also called locations); $q_0 \in Q$ is the initial state; $Inv : Q \rightarrow \Gamma(X)$ is an invariant assignment function; $L : Q \rightarrow \wp(AP)$ is a function labeling the states in Q with elements of AP; $T \subseteq_{fin} Q \times Q \times \Gamma(X) \times Act_\tau \times \wp(X)$ is a finite set of transitions.

Note that, to define the finite set of transitions T, we use relation $\subseteq_{fin}$ since the set of clock constraints $\Gamma(X)$—i.e., the universe of constraints that can be defined for a set of clocks—is infinite. A transition $t = (q, q', \gamma, \alpha, \zeta) \in T$ is written as $q \xrightarrow{\gamma, \alpha, \zeta} q'$; the notations t^-, t^+, t_g, t_e, t_s indicate, respectively, the source q, the destination q', the clock constraint γ, the event α and the set of clocks ζ to be reset when firing the transition. Fig. 2(a) shows a simple example of TA.

Let Int be a finite set of *integer variables* with values in $\mathbb{Z}$ and $\sim \in \{<, =\}$; $Assign(Int)$ is the set of assignments of the form $n := exp$, where $n \in Int$ and exp is an arithmetic expression over the integer variables and elements of $\mathbb{Z}$—i.e., exp is defined by the syntax $exp := exp + exp \mid exp - exp \mid exp \times exp \mid exp \div exp \mid n \mid c$, where $n \in Int$ and $c \in \mathbb{Z}$. $\Gamma(Int)$ is the set of *variable constraints* γ over Int defined as $\gamma := n \sim c \mid n \sim n' \mid \neg\gamma \mid \gamma \wedge \gamma$, where n and n' are integer variables and $c \in \mathbb{Z}$.



(a) An example of TA.



(b) An example of TA with Variables.

Fig. 2. The TA in (a) has three locations, q_0, q_1, q_2, and one clock x. The transition from q_2 to q_0 is labeled with guard $x = 10$. When the transition is taken, clock x is reset—i.e., it is set to 0. Location q_1 is associated with invariant $x \leq 5$. Locations q_0 and q_2 are labeled with atomic propositions a and c, respectively. The TA in (b) is the same as the one of (a), except for the presence of integer variable n, which is set to 0, 1 or 2 depending on the transition taken.

Definition 2.2 (TA with Variables). Let AP be a non-empty set of atomic propositions, X be a set of clocks, Act be a set of events and Int be a finite set of integer variables. A *Timed Automaton with Variables* is a tuple $\mathcal{A} = \langle AP, X, Act_\tau, Int, Q, q_0, v^0_{var}, Inv, L, T \rangle$, where: Q is a finite set of control states (also called locations); $q_0 \in Q$ is the initial state; $v^0_{\text{var}} : Int \rightarrow \mathbb{Z}$ assigns each variable with a value in $\mathbb{Z}$; $Inv : Q \rightarrow \Gamma(X)$ is an invariant assignment function; $L : Q \rightarrow \wp(AP)$ is a function labeling the states in Q with elements of AP; $T \subseteq_{fin} Q \times Q \times \Gamma(X) \times \Gamma(Int) \times Act_\tau \times \wp(X) \times \wp(Assign(Int))$ is a finite set of transitions.

A transition is written as $q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ where ξ is a constraint of $\Gamma(Int)$ and μ is a set of assignments from $\wp(Assign(Int))$. The notations t_d and t_u indicate, respectively, the variable constraint ξ and the set of assignments μ associated with a transition t. An example of TA with Variables is presented in Fig. 2(b).

Remark 1. *A set of assignments $\mu \in \wp(Assign(Int))$ might be inconsistent, i.e., it may assign different values to the same variable. For example, $\mu = \{x = 2, x = 3\}$ is inconsistent since two values are assigned to variable x. In this case, a transition associated with μ cannot be fired.*

Remark 2. *It is easy to see that TA with variables as defined in Definition 2.2 are undecidable, unless suitable restrictions are introduced. Indeed, as mentioned in Section 3, in this paper we consider only variables with finite domains. Note, however, that one of the goals of this paper is to present an encoding that (i) is general; (ii) considers different semantics of the (network of) TA; and (iii) can be implemented into a tool of practical usage. For this reason, we directly encode features such as variables with (finite) integer domains into CLTLoc, instead of relying on equivalent formulations of TA that only use a minimal number of constructs—e.g., by representing the value of variables with finite integer domains through suitable locations. Indeed, the current encoding can potentially be adopted even when infinite domains, combined with suitable constraints to ensure decidability, are considered (e.g., TA extended with reversal bounded integer counters, as in [17]).*

When networks of TA are considered, the event symbols labeling the transitions are used to synchronize automata. Every event symbol $\alpha \in Act$ is associated with one communication channel, which can be identified with the event symbol itself, i.e., channel α. To simplify the notation, even in the case of a network of TA we use Act_τ to indicate the set of actions of the network, which is redefined to include also the synchronization symbols. Specifically, when networks of TA are considered, the set of actions Act_τ is now defined as $Act_\tau = \{\tau\} \cup \{Act \times Sync\}$, where $Sync$ is a set of synchronization primitives and τ indicates that no synchronization primitive is associated with the transition. In this work, $Sync$ is restricted to $\{!, ?, \#, @, \&, *\}$ where the symbols ! and ? indicate that a TA emits and receives an event, respectively, # denotes a broadcast synchronization sender and @ denotes a broadcast synchronization receiver, & denotes a one-to-many synchronization sender and $*$ denotes a one-to-many communication receiver. Symbols $\alpha!$, $\alpha?$, $\alpha\#$, $\alpha@$, $\alpha\&$ and $\alpha*$ indicate the element $(\alpha, !)$, $(\alpha, ?)$, $(\alpha, \#)$, $(\alpha, @)$, $(\alpha, \&)$ and $(\alpha, *)$ such that $(\alpha, !)$, $(\alpha, ?)$, $(\alpha, \#)$, $(\alpha, @)$, $(\alpha, \&)$ and $(\alpha, *)$ is contained in the set $\{Act \times Sync\}$.

Definition 2.3 (Network of TA). A network $\mathcal{N}$ of TA is a set $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_K\}$ of TA defined over the same set of atomic propositions AP, actions Act_τ, variables Int and clocks X.

Remark 3. *Let $\mathcal{N}$ be a network of TA defined over the set of clocks X; a clock $x \in X$ is a* local clock *of an automaton $\mathcal{A}_i \in \mathcal{N}$ if x is used in the invariants, guards or resets of $\mathcal{A}_i$ and there is no other automaton $\mathcal{A}_j \in \mathcal{N}$, with $\mathcal{A}_i \neq \mathcal{A}_j$, using x in its invariants, guards or resets. Let Int be a set of variables of $\mathcal{N}$, a variable $n \in Int$ is a* local variable *of an automaton $\mathcal{A}_i \in \mathcal{N}$ if n is used in the guards or assignments of $\mathcal{A}_i$ and there is no other automaton $\mathcal{A}_j \in \mathcal{N}$, with $\mathcal{A}_i \neq \mathcal{A}_j$, using the variable n in its guards or assignments.*

2.2 Metric Interval Temporal Logic

An interval I is a convex subset of $\mathbb{R}_{\geq 0}$ of the form $\langle a, b\rangle$ or $\langle a, \infty)$, where $a \leq b$ are non-negative integers; symbol $\langle$ is either (or [; symbol $\rangle$ is either) or].

The syntax of (well-formed) MITL formulae is defined by the grammar

$$\phi := \alpha \mid \phi \wedge \phi \mid \neg\phi \mid \phi\, \mathcal{U}_I\, \phi$$

where α are atomic formulae. Since MITL is here used to specify properties of TA enriched with variables, atomic formulae α are either propositions of AP or formulae of the form $n \sim d$, where $n \in Int$, $d \in \mathbb{Z}$ and $\sim \in \{<, =\}$. In the following, set AP_v indicates the universe of the formulae of the form $n \sim d$.

The semantics of MITL is defined w.r.t. signals. Let $\mathbb{Z}^{Int}$ be the set of total functions from Int to $\mathbb{Z}$. A *signal* is a total function $M : \mathbb{R}_{\geq 0} \to \wp(AP) \times \mathbb{Z}^{Int}$. Let M be a signal; the semantics of an MITL formula is defined as follows.

$$\begin{array}{lll}
M, t \models p & \text{iff} & M(t) = (P, v_{\text{var}}) \text{ and } p \in P \\
M, t \models n \sim d & \text{iff} & M(t) = (P, v_{\text{var}}) \text{ and } v_{\text{var}}(n) \sim d \\
M, t \models \neg p & \text{iff} & M, t \models \neg\phi \\
M, t \models \phi \wedge \psi & \text{iff} & M, t \models \phi \text{ and } M, t \models \psi \\
M, t \models \phi\, \mathcal{U}_I\, \psi & \text{iff} & \exists t' > t, t' - t \in I, M, t' \models \psi \text{ and } \forall t'' \in (t, t'),\ M, t'' \models \phi
\end{array}$$

An MITL formula ϕ is *satisfiable* if there exists a signal M, such that $M, 0 \models \phi$. In this case, M is called a *model* of ϕ.

2.3 Constraint LTL over clocks

CLTLoc is a temporal logic where formulae are defined over a finite set of atomic propositions and a set of dense variables over $\mathbb{R}_{\geq 0}$ representing clocks. CLTLoc has been recently extended by supporting expressions over a set of arithmetical variables [29]. CLTLoc is the intermediate language that is adopted to solve the model-checking problem of TA with MITL specifications.

CLTLoc allows for two kinds of atomic formulae: over clocks and over arithmetical variables. An atomic formula over a clock x is for instance $x < 4$, whereas an atomic formula over arithmetical variables is for example $n+m < 4$, with $n, m \in \mathbb{Z}$. Similarly to TA, a clock x measures the time elapsed since its last "reset". CLTLoc also exploits the "next" $\mathcal{X}$ modality applied to integer variables [19]: if n is an integer variable, the term $\mathcal{X}(n)$ represents the value of n in the next position.

Let X be a finite set of clocks and *Int* be a finite set of integer variables; formulae of CLTLoc with counters are defined by the grammar:

$$\phi := p \mid x \sim c \mid exp_1 \sim exp_2 \mid \mathcal{X}(n) \sim exp \mid \phi \wedge \phi \mid \neg\phi \mid \mathcal{X}\,\phi \mid \phi\,\mathcal{U}\,\phi$$

where $p \in AP$, $c \in \mathbb{N}$, $x \in X$, exp, exp_1 and exp_2 are arithmetic expressions over the sets Int and $\mathbb{Z}$ (defined as described in Section 2.1), $n \in Int$ and $\sim$ is a relation in $\{<, =\}$. $\mathcal{X}$, $\mathcal{U}$ are the usual "next" and "until" operators of LTL. Modalities such as "eventually" ($\mathcal{F}$), "globally" ($\mathcal{G}$), and "release" ($\mathcal{R}$) are defined as usual. Symbol $\top$ (true) abbreviates $(p \vee \neg p)$, for some $p \in AP$.

The strict linear order $(\mathbb{N}, <)$ is the standard representation of positions in time. The interpretation of *clocks* is defined by means of a clock valuation $\sigma : \mathbb{N} \times X \longrightarrow \mathbb{R}_{\geq 0}$ assigning, for every position $i \in \mathbb{N}$, a real value $\sigma(i, x)$ to each clock $x \in X$. As in TA, a clock x measures the time elapsed since the last time when $x = 0$, i.e., the last "reset" of x. The semantics of time evolution is *strict*, namely the value of a clock must strictly increase in two adjacent time positions, unless it is reset (i.e., for all $i \in \mathbb{N}$, $x \in X$, it holds that $\sigma(i+1, x) > \sigma(i, x)$, unless $\sigma(i+1, x) = 0$ holds)[1]. To ensure that time strictly progresses at the same rate for every clock, σ must satisfy the following condition: for every position $i \in \mathbb{N}$, there exists a "time delay" $\delta_i > 0$ such that for every clock $x \in X$:

$$\sigma(i+1, x) = \begin{cases} \sigma(i, x) + \delta_i & \text{progress} \\ 0 & \text{reset } x \end{cases}$$

If this is the case, then σ is called a *clock assignment*. The initial value $\sigma(0, x)$ may be any non-negative value. Moreover, a clock assignment is such that $\sum_{i \in \mathbb{N}} \delta_i = \infty$, i.e., time is always progressing.

The interpretation of *variables* is defined by a mapping $\iota : \mathbb{N} \times Int \longrightarrow \mathbb{Z}$ assigning, for every position $i \in \mathbb{N}$, a value in $\mathbb{Z}$ to each variable of set Int. Let ι be a valuation and i be a position; $exp(\iota, i)$ indicates the evaluation of exp obtained by replacing each arithmetical variable $n \in Int$ that occurs in exp with value $\iota(i, n)$. An interpretation of CLTLoc is a triple (π, σ, ι), where $\pi : \mathbb{N} \longrightarrow \wp(AP)$ is a mapping associating a set of propositions with each position $i \in \mathbb{N}$, σ is a clock assignment and ι is a valuation of variables. Let x be a clock, n be a variable and c be a constant in $\mathbb{N}$, the semantic of CLTLoc at a position $i \in \mathbb{N}$ over an interpretation (π, σ, ι) is defined as follows (standard LTL modalities are omitted):

$$\begin{array}{lll}
(\pi, \sigma, \iota), i \models x \sim c & \text{iff} & \sigma(i, x) \sim c \\
(\pi, \sigma, \iota), i \models exp_1 \sim exp_2 & \text{iff} & exp_1(\iota, i) \sim exp_2(\iota, i) \\
(\pi, \sigma, \iota), i \models \mathcal{X}(n) \sim exp & \text{iff} & \iota(i+1, n) \sim exp(\iota, i) \\
(\pi, \sigma, \iota), i \models a & \text{iff} & a \in \pi(i)
\end{array}$$

A CLTLoc formula ϕ is *satisfiable* if there exist an interpretation (π, σ, ι) such that $(\pi, \sigma, \iota), 0 \models \phi$. In this case, (π, σ, ι) is called a *model* of ϕ, written $(\pi, \sigma, \iota) \models \phi$. It is easy to see that CLTLoc is undecidable, as it can encode a 2-counter machine; however, in this work a decidable subset of CLTLoc is adopted, where the domain of arithmetical variables is *finite*.

[1] As discussed in the following this assumption does not allow us to capture the super-dense semantics of TAs.

3 CONTINUOUS TIME SEMANTICS FOR TIMED AUTOMATA

The behavior of TA over time is described by means of execution traces that define the evolution of the APs, variables and clocks of the automata changing their values because transitions are taken or because time elapses. When networks of synchronizing TA are considered, the formal definition of the semantics of (network of) TA has to deal with the following issues:

a) how the automata progress over time by means of transitions associated with actions (liveness conditions); and

b) how the automata synchronize when transitions labeled with !,?, #, @, & and $*$ are fired.

Only the general case semantics of a network of TA with variables is discussed hereafter. Obviously, the semantics of a network of TA without variables or of a single TA are just special cases. Furthermore, in the rest of this paper, integer variables are restricted to finite domains.

3.1 Preliminaries

Let X be a set of clocks and $\gamma \in \Gamma(X)$ be a clock constraint. A *clock valuation* is a function $v : X \rightarrow \mathbb{R}_{\geq 0}$; the notation $v \models \gamma$ indicates that the clock valuation v satisfies γ—i.e., by replacing $v(x)$ for x in any subformula of the form $x \sim d$ the clock constraint γ evaluates to true. Let r be an element of $\mathbb{R}$, $v + r$ denotes the clock valuation mapping clock x to value $v(x) + r$—i.e., $(v + r)(x) = v(x) + r$ for all $x \in X$. In the following, without loss of generality, the clock constraints associated with transitions of TA are supposed to define convex sets of $\mathbb{R}$. Every transition in a TA whose guard is non-convex can be replaced by at most an exponential number of fresh transitions, each one labeled with a convex guard. In fact, any non-convex sets of $\mathbb{R}$ defined by a clock constraint can be defined as the union of finitely many convex ones. For this reason, in the next sections, clock constraints are restricted to conjunctions of atomic clock constraints of the form $x \sim d$, where $\sim$ is a relation in $\{<, =, >, \leq, \geq\}$.[2]

A transition from q to q' labeled with a non-convex guard can be equivalently replaced with a set of transitions, all starting in q and leading to q', labeled with convex guards.

Before providing the formal definition of the transition relation for networks of TA, the notion of *weak* satisfaction relation $\models_w$ over clock valuations and clock constraints is introduced. The weak satisfaction relation may be used to evaluate the invariants in the locations when a transition is fired, to allow different ways of performing an instantaneous transition. In particular, relation $\models_w$ is never satisfied when a clock constraint has the form $x = d$, where x is a clock and d is a positive integer. The importance of weak satisfaction will be clearer later. Relation $\models_w$ weakens the evaluation of clock constraints of the form $x < d$ and $x > d$. Those two constraints are weakly satisfied for $v(x) = d - \epsilon$ or $v(x) = d + \epsilon$ (where $\epsilon > 0$) as in the non-weak case, but they are weakly satisfied also for $v(x) = d$ (i.e., as if the constraints were of the forms $x \leq d$ and $x \geq d$). Formally, a clock valuation v weakly satisfies a clock constraint γ, written $v \models_w \gamma$, when the following conditions hold:

$$\begin{array}{lll} v \models_w x \sim d & \text{iff} \quad v(x) \sim d \text{ or } v(x) = d & \sim \in \{<, >, \leq, \geq\} \\ v \not\models_w x = d & \text{for any } x \in X, d \in \mathbb{N} & \end{array}$$

Naturally, $\models_w$ can be extended to conjunctions of formulae $x \sim d$. For instance, the formula $x < 1 \wedge y > 1$ is both satisfied and weakly satisfied by the clock evaluation such that $v(x) = 0.8$ and $v(y) = 1.2$, but it is only weakly satisfied if $v(x) = 1$ and $v(y) = 1$.

A *variable valuation* v_{var} is a function $v_{\text{var}} : \mathit{Int} \rightarrow \mathbb{Z}$ that maps each variable in *Int* to an integer number; also, if $\xi \in \Gamma(\mathit{Int})$ is a variable constraint, $v_{\text{var}} \models \xi$ indicates that valuation v_{var} satisfies ξ.

[2]Relations $\leq, \geq, >$ can obviously be obtained by combining atomic clock constraints $x \sim d$ (with $\sim \in \{<, =\}$) and the negation operator ($\neg$).

Let $t = q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ be a transition, v be a clock valuation and v_{var} be a variable valuation, t is *enabled* in the valuation when v satisfies γ and v_{var} satisfies ξ. In addition, a satisfaction relation for assignments is here introduced. Let v_{var} and v'_{var} be two variable valuations; $(v'_{\text{var}}, v_{\text{var}}) \models \mu$ indicates that all the assignments in μ are satisfied by means of v'_{var} and v_{var}. Formally, all assignments of the form $n = exp$ hold when n is replaced with $v'_{\text{var}}(n)$ and every occurrence of $m \in Int$ in exp is replaced with $v_{\text{var}}(m)$. Moreover, let $U(\mu)$ be the set of variables that are updated by μ—that is, that appear as the left-hand side in an assignment of μ—and let $U(t)$ indicate the set $U(\mu)$ given a transition t.

Remark 4. *Relation $\models$ does not hold for inconsistent transitions, i.e., assigning multiple distinct values to a variable. For example, if $\mu = \{n = 2, n = 3\}$, there is no assignment to n such that both $n = 2$ and $n = 3$ hold.*

Definition 3.1. Let $\mathcal{N}$ be a network of K TA. A *configuration* of $\mathcal{N}$ is a tuple $(\mathtt{l}, v_{\text{var}}, v)$ where $\mathtt{l}$ is a vector $[q^1, \dots, q^K]$— s.t. $q^1, \dots, q^K$ are locations of $\mathcal{A}_1, \dots, \mathcal{A}_K$—$v_{\text{var}}$ (resp., v) is a variable (resp., clock) valuation for the set Int (resp., X) including all integer variables (resp., clocks) appearing in the TA of the network.

When a network of TA is considered, it is possible that some automata in the network take a transition while the remaining others do not fire a transition and keep their state unchanged. Firing a transition labeled with the null event τ (i.e., a transition that does not synchronize) is however different from not taking a transition at all. Symbol _ indicates that an automaton k does not perform any transition in T_k.

The notation $\mathtt{l}[k]$ indicates the location of automaton $\mathcal{A}_k$—i.e., if $\mathtt{l}[k] = j$, then automaton $\mathcal{A}_k$ is in location q^k_j, assuming that the locations of each automaton are numbered, with 0 indicating the initial one.

Two kinds of configuration changes may occur when an automaton in the network performs a transition from a state q to q'. They are indicated in Def. 3.2 with symbols `ei` (excluded-included) and `ie` (included-excluded). Intuitively, these symbols constraints how the network behaves when a transition is fired. The symbol `ei` forbids an automaton to be in state q (excluded) in the instant in which the transition from q to q' is fired, while it forces the automaton to be in state q' (included). Vice versa, the symbol `ie` forces an automaton to be in state q (included) in the instant in which the transition from q to q' is fired, while it forbids the automaton to be in state q' (excluded). Consider, for instance, a location q labeled with $x < 1$ and an outgoing transition. If the automaton is in q, then the transition can be executed when $v(x) < 1$, in which case the corresponding configuration change can be arbitrarily marked either with `ei` or with `ie`. The transition can be executed even when $v(x) = 1$ holds, but in this case the kind of configuration change can only be `ei`.

Definition 3.2. Let $\mathcal{N}$ be a network of K TA. Let $(\mathtt{l}, v_{\text{var}}, v)$, $(\mathtt{l}', v'_{\text{var}}, v')$ be two configurations, let $\delta \in \mathbb{R}_{>0}$ and Λ be a tuple of K symbols such that $\Lambda[k] \in \{Act_\tau \times \{\mathtt{ei}, \mathtt{ie}\}\} \cup \{\_\}$ for every $1 \le k \le K$. Then, a configuration change is either a transition $(\mathtt{l}, v_{\text{var}}, v) \xrightarrow{\Lambda} (\mathtt{l}', v'_{\text{var}}, v')$ or a transition $(\mathtt{l}, v_{\text{var}}, v) \xrightarrow{\delta} (\mathtt{l}', v'_{\text{var}}, v')$ defined as follows.

(1) $(\mathtt{l}, v_{\text{var}}, v) \xrightarrow{\Lambda} (\mathtt{l}', v'_{\text{var}}, v')$ occurs if
  (a) for each $\Lambda[k] = (\alpha, b)$ there is a transition $l[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} l'[k]$ in $\mathcal{A}_k$ such that:
    (i) $v \models \gamma$ and $v_{\text{var}} \models \xi$,
    (ii) $v'(x) = 0$ holds for all $x \in \zeta$,
    (iii) $(v'_{\text{var}}, v_{\text{var}}) \models \mu$,
    (iv) when $b = \mathtt{ei}$ then:

- $v \models_w Inv(\mathtt{l}[k])$ and
- $v' \models Inv(\mathtt{l}'[k])$

(v) when $b = \mathtt{ie}$:
- $v \models Inv(\mathtt{l}[k])$ and
- $v' \models_w Inv(\mathtt{l}'[k])$

(b) for each $\Lambda[k] = \_$ it holds that:
  (i) $\mathtt{l}'[k] = \mathtt{l}[k]$;
  (ii) $v \models Inv(\mathtt{l}[k])$ and $v' \models Inv(\mathtt{l}'[k])$.

(c) for each clock $x \in X$ (resp., integer variable $n \in Int$), if x (resp., n) does not appear in any ζ (resp., it is not assigned by any A) of one of the transitions taken by $\mathcal{A}_1, \dots, \mathcal{A}_K$, then $v'(x) = v(x)$ (resp., $v'_{\text{var}}(n) = v_{\text{var}}(n)$);

(2) $(\mathtt{l}, v_{\text{var}}, v) \xrightarrow{\delta} (\mathtt{l}', v'_{\text{var}}, v')$ occurs if $\mathtt{l}'[k] = \mathtt{l}[k]$, $v'_{\text{var}} = v_{\text{var}}$, $v' = v + \delta$ and for all $1 \leq k \leq K$, $v' \models_w Inv(\mathtt{l}[k])$.

A configuration change $(\mathtt{l}, v_{\text{var}}, v) \xrightarrow{\Lambda} (\mathtt{l}', v'_{\text{var}}, v')$, for some $\Lambda \in \{\{Act_\tau \times \{\mathtt{ei}, \mathtt{ie}\}\} \cup \{\_\}\}^K$, satisfying (1) is called a *discrete transition*. If it satisfies (2) then it is called a *time transition*. For convenience of notation, symbols $(\alpha, \mathtt{ei})$ and $(\alpha, \mathtt{ie})$, for some $\alpha \in Act_\tau$, are hereinafter denoted respectively with $\alpha^{)[}$ and $\alpha^{](}$, meaning that the discrete transition performed by the k-th automaton, such that $\Lambda[k] = (\alpha, \mathtt{ei})$ (resp., $\Lambda[k] = (\alpha, \mathtt{ie})$), is *open-closed* (resp., *closed-open*). The use of symbols $\alpha^{)[}$ or $\alpha^{](}$ allows the distinction of two different ways of performing a transition by means of an action α. The two modes are determined by the conditions in (1)(a)iv and (1)(a)v and depend on the invariants of the locations involved in the transition, the clock values and the resets applied in the configuration change. Location invariants and resets make it possible to constrain every symbol $\Lambda[k]$, associated with $\mathcal{A}_k$, and hence to define how the configuration change in $\mathcal{A}_k$ is realized. Cases (1) and (2) are discussed in detail in the following.

Case (1). If the discrete transition is open-closed—i.e., the symbol is $\alpha^{)[}$—then (1)(a)iv holds. The conditions of this case impose that v' satisfies the invariant of the destination location and v *weakly* satisfies the invariant of the source location. Therefore, if the invariant of the source state is $x < 1$, then the transition can be taken with $v(x) \leq 1$. This is achieved through the weak satisfaction relation that guarantees the (weak) satisfaction of the invariant $x < 1$ with $v(x) = 1$. Conversely, if the discrete transition is closed-open—i.e., the symbol is $\alpha^{](}$—then (1)(a)v holds. The conditions defined therein allow the invariant of the destination location to be weakly satisfied and the transition to be fired with $v(x) \geq 1$, if the invariant of the destination state is $x > 1$.

Based on the invariants and resets, the symbol $\Lambda[k]$ is either non-deterministically chosen between $\alpha^{)[}$ and $\alpha^{](}$ because both symbols are allowed, or it is determistically defined because only one is permitted, according to conditions (1)(a)iv and (1)(a)v. Figure 3 shows two automata and the possible transitions. The case of a transition on a symbol $\alpha^{)[}$ is exemplified in Fig. 4(a). When $v(x) \leq 1$ holds, the invariant $Inv(\mathtt{l}[k])$ of the first location is weakly satisfied by v—i.e., $v \models_w Inv(\mathtt{l}[k])$ holds. In such a case, since $Inv(\mathtt{l}'[k])$ is empty—hence it is trivially true—symbol $\alpha^{)[}$ is allowed. In the automaton of Fig. 4(b), instead, only symbol $\alpha^{](}$ is allowed, because the constraint on the second location requires that x is not reset. Constraint $v' \models_w Inv(\mathtt{l}'[k])$ of condition (1)(a)v is satisfiable with $v'(x) = 0$. Conversely, $v \models Inv(\mathtt{l}[k])$ of condition (1)(a)iv would be falsified because of the reset and, hence, $\alpha^{)[}$ is prevented. In addition, $\alpha^{](}$ is also allowed in the automaton of Fig. 4(a) when $v(x) < 1$ holds, because $v \models Inv(\mathtt{l}[k])$ holds in that case.

Case (2). It defines the time transitions by means of the weak relation $\models_w$. Consider an open-closed transition that changes the location of an automaton currently in q, for some q, and the last time transition immediately preceding it which makes the time progress of δ time units, for some

$\delta > 0$. In order to perform the open-closed transition, $v + \delta$ weakly satisfies $Inv(q)$, as required by (1)(a)iv. The use of relation $\models$ instead of $\models_w$ in (2) would prevent the occurrence of some open-closed configuration changes, i.e., those that would be caused by an $Inv(q)$ being weakly, but not strongly, satisfied in the time transition. In case of closed-open transitions, $v + \delta$ (strongly) satisfies $Inv(q)$, as required by (1)(a)v. Hence, $v + \delta$ also weakly satisfies $Inv(q)$, as in condition (2). For instance, in Fig. 4(a), if the automaton is in the location labeled with $x < 1$ and $v(x) = 0.8$ then the time progress $\delta = 0.2$ is permitted by (2) in order to perform the outgoing transition in an open-closed manner with $v'(x) = 1$. Moreover, if the time progress δ is such that the invariant of the current location q is such that $v + \delta \models Inv(q)$ holds, then both kinds of transitions are allowed.

The combination of conditions (1) and (2) describe how the configuration of a network of TA changes.

Based on the previous arguments, the symbols $\alpha^{)[}$ and $\alpha^{](}$ will be considered in Sec. 5 to define the signals associated with atomic propositions and variables when discrete transitions are taken.

The notion of trace is now introduced. Recall that, $v^0_{\text{var}} : Int \rightarrow \mathbb{Z}$ assigns each variable with a value in $\mathbb{Z}$ (see Def 2.2) .

Definition 3.3. Let $\mathcal{N}$ be a network of K TA. A *trace* is an infinite sequence

$$\eta = (\mathtt{l}_0, v_{\text{var},0}, v_0), e_0, (\mathtt{l}_1, v_{\text{var},1}, v_1), e_1, (\mathtt{l}_2, v_{\text{var},2}, v_2), e_2, \ldots$$

such that:

(1) for all $i \in \mathbb{N}$, $e_i = \Lambda_i$ or $e_i = \delta_i$;
(2) for all $i \in \mathbb{N}$ it holds that $(\mathtt{l}_i, v_{\text{var},i}, v_i) \xrightarrow{e_i} (\mathtt{l}_{i+1}, v_{\text{var},i+1}, v_{i+1})$;
(3) $e_0 = \delta_0$, for some $\delta_0 \in \mathbb{R}_{>0}$;
(4) for all $1 \leq k \leq K$, it holds that $\mathtt{l}_0[k] = 0$, $v_0 \models Inv(\mathtt{l}_0[k])$, for all $x \in X$ it holds that $v_0(x) = 0$, and for all $n \in Int$ it holds that $v_{\text{var},0}(n) = v^0_{\text{var}}(n)$.
(5) discrete transitions must be followed by time transitions; that is, if e_i is a discrete transition ($e_i = \Lambda_i$), then e_{i+1} is a time transition ($e_i = \Lambda_i$).

The word $w(\eta)$ of a trace η is the sequence $e_0 e_1 \ldots$.

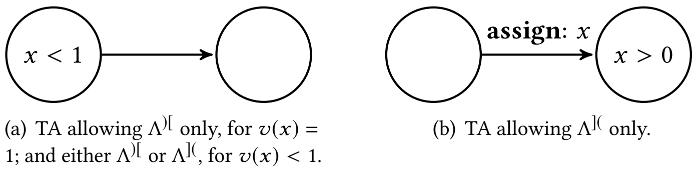
With a slight abuse of notation, a trace $(\mathtt{l}_0, v_{\text{var},0}, v_0), e_0, (\mathtt{l}_1, v_{\text{var},1}, v_1), e_1, (\mathtt{l}_2, v_{\text{var},2}, v_2), e_2, \ldots$ can be written as $(\mathtt{l}_0, v_{\text{var},0}, v_0) \xrightarrow{e_0} (\mathtt{l}_1, v_{\text{var},1}, v_1) \xrightarrow{e_1} (\mathtt{l}_2, v_{\text{var},2}, v_2) \xrightarrow{e_2} \ldots$.

Notice that case (2) of Definition 3.2 does not impose that, when a transition $(\mathtt{l}_i, v_{\text{var},i}, v_i) \xrightarrow{\delta_i} (\mathtt{l}_{i+1}, v_{\text{var},i+1}, v_{i+1})$ is taken, the invariant hold at the beginning of the interval of length δ (i.e., for valuation v_i), but only at its end (i.e., for valuation v_{i+1}). Indeed, transitions are meaningful only in the context of traces, and the fact that the invariant holds for v_i is guaranteed by condition (4) of Definition 3.3 if $i = 0$, otherwise by the definition of transition $(\mathtt{l}_{i-1}, v_{\text{var},i-1}, v_{i-1}) \xrightarrow{e_{i-1}} (\mathtt{l}_i, v_{\text{var},i}, v_i)$ in the trace (no matter the nature of e_{i-1}).

Since by condition (5) there cannot be two consecutive discrete transitions, and since any finite sequence of consecutive delays $\delta_h \ldots \delta_{h+k}$, with $k \geq 0$, is equivalent to a single delay $\sum_{i=h}^{h+k} \delta_i$, a trace can always be rewritten into a new one such that discrete and time transitions strictly alternate. Moreover, by the previous property, every time transition δ_h can always be replaced with a finite sequence of m pairs of time and discrete transition $\delta_{h,0}\Lambda_{h,0}\delta_{h,1}\Lambda_{h,2} \ldots \delta_{h,m}$, strictly alternating, such that in $\Lambda_{h,i}[k] = \_$, for all $0 \leq i \leq m - 1$ and $\delta_h = \sum_{i=0}^{m} \delta_{h,i}$. This property is used in Sec. 5 to allow the use of [9] in the resolution of the model-checking problem of TA with MITL.

To facilitate future discussions, a trace is represented with the following notation where the numbering of configurations increases only after the discrete transitions:

Fig. 3. Two examples of transitions enforcing a different and unique configuration change.



(a) TA allowing $\Lambda^{)[}$ only, for $v(x) = 1$; and either $\Lambda^{)[}$ or $\Lambda^{](}$, for $v(x) < 1$.

(b) TA allowing $\Lambda^{](}$ only.

$$(\mathtt{l}_0, v_{\text{var},0}, v_0) \xrightarrow{\delta_0} (\mathtt{l}_0', v_{\text{var},0}', v_0') \xrightarrow{\Lambda_0} (\mathtt{l}_1, v_{\text{var},1}, v_1) \xrightarrow{\delta_1} \dots$$

3.2 Liveness and synchronization

Definition 3.3 only provides weak conditions on the occurrences of discrete transitions and does not express any restriction on how TA synchronize. In fact, beside the first three conditions requiring that traces are sequences of configuration changes starting from a specific initial configuration, only Condition (5) expresses a restriction on how the configuration changes occur, which only prevents discrete transitions from occurring consecutively, one after the other. However, one is typically interested in "live" traces, in which some transition is eventually taken and where the effect of the synchronizing primitives is precisely defined.

3.2.1 Liveness. Table 1 shows the formal definition of the following four possible liveness conditions, for a generic trace of a network with K timed automata.

- *Strong (Weak) transition liveness:* at any time instant, *each* (resp., *at least one*) automaton of the network eventually performs a transition.
- *Strong (Weak) guard liveness:* at any time instant, *for each automaton* (resp., *there exists an automaton* such that) the values of clocks and variables will eventually enable one of its transitions[3].

Even if the previous conditions restrict the occurrence of transitions or the satisfiability of guards along the trace, they do not prevent, in general, the progress of time from slowing down. This issue is well-known in the literature of timed verification and, intuitively, it is caused by so-called *time convergent* traces, where the sum of all the delays δ_i associated with time transitions is bounded by some positive integer. Therefore, the previous liveness conditions allow Zeno traces, i.e., where infinitely many actions can occur in finite time. Avoiding Zeno traces can be done in several ways. For instance, one can require strong transition liveness and introduce a new TA in the network which infinitely many times along the trace resets a clock when the clock value reaches 1.

3.2.2 Synchronization. Section 2 introduced qualifiers !, ?, #, @, &, and $*$ labeling actions on the transitions with the goal of capturing different ways in which the automata of a network can synchronize. Qualifiers ! and ? describe a so-called *channel-based* synchronization; qualifiers # and @ describe a *broadcast* synchronization; and qualifiers & and $*$ describe a *one-to-many* synchronization.

Channel-based, broadcast and one-to-many synchronizations can be arbitrarily mixed in the same configuration change. Table 2 shows the formal definition of the channel-based, broadcast and one-to-many synchronization mechanisms for a generic trace of a network of TA.

[3]The constraint does not force the transition to be taken. Moreover, alternative definitions can be given by considering only clock or variable guards.

Table 1. Formal definition of different liveness properties for traces.

Name	**Formulation of the Semantics**
Strong transition liveness	For every $h \geq 0$ and $1 \leq k \leq K$, there exists $j > h$ such that $(\mathrm{l}'_j, v'_{\mathrm{var},j}, v'_j) \xrightarrow{\Lambda_j} (\mathrm{l}_{j+1}, v_{\mathrm{var},j+1}, v_{j+1})$ belongs to the trace and $\Lambda_j[k] \neq \_$.
Weak transition liveness	For every $h \geq 0$ there exist $1 \leq k \leq K$ and $j > h$ such that $(\mathrm{l}'_j, v'_{\mathrm{var},j}, v'_j) \xrightarrow{\Lambda_j} (\mathrm{l}_{j+1}, v_{\mathrm{var},j+1}, v_{j+1})$ belongs to the trace and $\Lambda_j[k] \neq \_$.
Strong guard liveness	For every $h \geq 0$ and $1 \leq k \leq K$, there exist $j > h$ and a configuration $(\mathrm{l}'_j, v'_{\mathrm{var},j}, v'_j)$ in the trace such that there is a transition $q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ with $q = \mathrm{l}'_j[k]$, for which $v_{\mathrm{var},j} \models \xi$ and $v_j \models \gamma$ hold.
Weak guard liveness	For every $h \geq 0$ there exist $1 \leq k \leq K$, $j > h$, and a configuration $(\mathrm{l}'_j, v'_{\mathrm{var},j}, v'_j)$ in the trace such that there is a transition $q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ with $q = \mathrm{l}'_j[k]$, for which $v_{\mathrm{var},j} \models \xi$ and $v_j \models \gamma$ hold.

- *Channel-based synchronization:* for any discrete transition, every "sending" (qualifier !) action in a TA $\mathcal{A}_k$ must be matched by exactly one corresponding "receiving" (qualifier ?) action in another TA $\mathcal{A}_{k'}$ on the same channel (e.g., $\alpha!$ and $\alpha?$) labeling an enabled transition.
- *Broadcast synchronization:* for any discrete transition, every "sending" (qualifier #) action in a TA $\mathcal{A}_k$ is matched by all and only "receiving" (qualifier @) actions in TA $\mathcal{A}_{k'}$ labeling an enabled transition. In other words, either $\mathcal{A}_{k'} \neq \mathcal{A}_k$ takes a transition labeled with $\alpha@$, or it does not exist any enabled transition for $\mathcal{A}_{k'}$ labeled with $\alpha@$.
- *One-to-many synchronization:* for any discrete transition, every "sending" (qualifier &) action in $\mathcal{A}_k$ is matched by a (non empty) set of "receiving" (qualifier $*$) actions in $\mathcal{A}_{k'}$ labeling an enabled transition. The one-to-many synchronization is a variation of the broadcast in which when an automaton $\mathcal{A}_k$ sends a message it is received by at least one receiver. However, not all the automata that have a transition labeled with $\alpha*$ are forced to receive the message.

Notice that, for any channel α, the previous synchronizations allow only one TA to send a message on α at any time instant; on the other hand, distinct TA can send messages concurrently on separate channels.

4 FROM TIMED AUTOMATA TO CLTLOC

This section shows that, given a network $\mathcal{N}$ of TA, it is possible to construct a CLTLoc formula $\Phi_{\mathcal{N}} = \varphi_{\mathcal{N}} \wedge \varphi_l \wedge \varphi_s \wedge \varphi_{ef}$ whose models represent the traces of $\mathcal{N}$. Formulae $\varphi_{\mathcal{N}}$, φ_l, φ_s and φ_{ef} encode, respectively: the behavior of the network, that is the effect of the transitions on the configuration of the network including how clocks are reset, how variables and locations are modified and when transitions can be taken ($\varphi_{\mathcal{N}}$); a set of constraints on the liveness conditions (φ_l); the semantics of the firing of the transitions that depend on the synchronization modifiers that decorate their labeling events (φ_s); and constraints on the types of edges (open-closed or closed-open) with which transitions are taken (φ_{ef}). Before discussing CLTLoc formulae $\varphi_{\mathcal{N}}$, φ_l φ_s and φ_{ef}, formula ϕ_{clock} is first introduced to encode a set of constraints on the CLTLoc clocks used in $\varphi_{\mathcal{N}}$.

Table 2. Definition of different constraints on traces depending on synchronization primitives.

Type	**Formulation of the Semantics**
Channels	For every configuration change $(1'_h, v'_{\text{var},h}, v'_h) \xrightarrow{\Lambda_h} (1_{h+1}, v_{\text{var},h+1}, v_{h+1})$ and $1 \leq k \leq K$ such that $\alpha! = \Lambda_h[k]$ holds, there exists exactly one $1 \leq k' \leq K$ such that $k' \neq k$ and $\alpha? = \Lambda_h[k']$ hold, and vice-versa.
Broadcast	For every configuration change $(1'_h, v'_{\text{var},h}, v'_h) \xrightarrow{\Lambda_h} (1_{h+1}, v_{\text{var},h+1}, v_{h+1})$ and $1 \leq k \leq K$ such that $\alpha\# = \Lambda_h[k]$ holds, for every $1 \leq k' \leq K$, with $k' \neq k$, it holds that $\alpha\# \neq \Lambda_h[k']$ and either $\alpha@ = \Lambda_h[k']$ holds, or no transition $q \xrightarrow{\gamma,\xi,\alpha@,\zeta,\mu} q'$ in $\mathcal{A}_{k'}$ is such that $v'_{\text{var},h} \models \xi$ and $v'_h \models \gamma$ hold.
One-to-many	For every configuration change $(1'_h, v'_{\text{var},h}, v'_h) \xrightarrow{\Lambda_h} (1_{h+1}, v_{\text{var},h+1}, v_{h+1})$ and $1 \leq k \leq K$ such that $\alpha\& = \Lambda_h[k]$, there exists at least one $1 \leq k' \leq K$ such that $k' \neq k$ and $\alpha* = \Lambda_h[k']$ hold.

Table 3. Encoding of the clocks of the automata.

$\phi_1 := \bigwedge_{x \in X}(x_0 = 0 \ \land \ x_1 > 0 \ \land \ \mathsf{x}_v = 0)$
$\phi_2(j) := \bigwedge_{x \in X}(x_j = 0) \rightarrow \mathcal{X}((x_{(j+1) \mod 2} = 0)\,\mathcal{R}((\mathsf{x}_v = j) \land (x_j > 0)))$

4.1 Encoding constraints over clocks (ϕ_{clock})

Unlike those in TA, clocks in CLTLoc formulae cannot be tested and reset at the same time. For instance, while it is possible that a transition in a TA both has guard $x > 5$ and resets clock x, in CLTLoc simultaneous test and reset would yield a contradiction, as testing $x > 5$ and resetting x in the same position equals to formula $x > 5 \land x = 0$. Therefore, for each clock $x \in X$, two clocks x_0 and x_1 are introduced in $\Phi_{\mathcal{N}}$ to represent a single clock x of the automaton. An additional Boolean variable x_v keeps track, in any discrete time position, of which clock between x_0 and x_1 is "active" (x_v being equal to 0 or 1 respectively). Clocks x_0 and x_1 are never reset at the same time and their resets alternate. If $\mathsf{x}_v = 0$ (resp., $\mathsf{x}_v = 1$) at position i of the model of $\Phi_{\mathcal{N}}$, then x_0 (resp., x_1) is the active clock at i and $\sigma(i, x_0)$ (resp., $\sigma(i, x_1)$) is the value used to evaluate the clock constraints at i. If the reset of x has to be represented at i, clock x_1 (resp., x_0) is set to 0 and the value x_v in position $i + 1$ is set to 1 (resp., 0)—i.e., the active clock is switched.

Table 3 shows the formulae ϕ_1 and ϕ_2 that are used to define ϕ_{clock}. Formula ϕ_1 specifies that initially, the active clock is x_0. In position 0 variable x_v is equal to 0 (indicating that x_0 is the active clock), x_0 is also equal to 0 and x_1 has an arbitrary value greater than zero. Formula ϕ_2 specifies that if x_j is reset, it cannot be reset again before $x_{(j+1) \mod 2}$ is reset. For instance, if clock x_0 is reset, then it cannot be reset again (it remains grater than zero) and it is the active clock ($\mathsf{x}_v = 0$) as long as x_1 is different from 0.

Formula ϕ_{clock} is defined as $\phi_1 \land \mathcal{G}(\phi_2(0) \land \phi_2(1))$. Since every clock x is represented by two variables x_0 and x_1, all clock constraints of the form $x \sim d$ in $\Gamma(X)$ that appear in the automaton are translated by means of the following CLTLoc formula (notice that a CLTLoc clock constraint of the form $x_i > d$ is an abbreviation for $\neg(x_i < d \lor x_i = d)$, and so on):

$$\phi_{x \sim d} := ((x_0 \sim d) \land (\mathsf{x}_v = 0)) \lor ((x_1 \sim d) \land (\mathsf{x}_v = 1)).$$

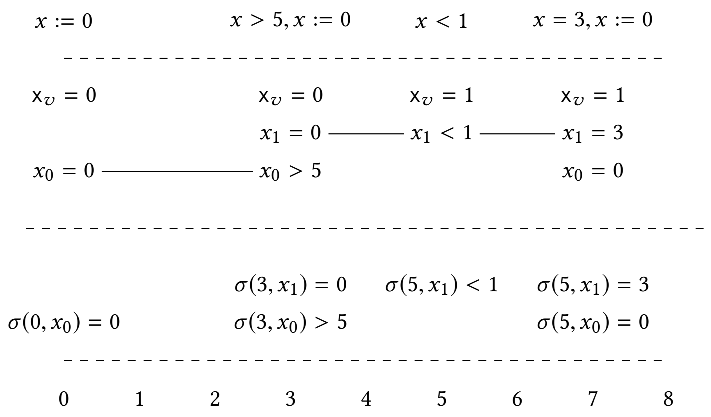


Fig. 4. Representation of tests and resets of clock x by means of the two copies x_0 and x_1.

Example 4.1. Figure 4 depicts a sequence of tests and resets of clock x over 9 discrete positions. The first row shows the sequence of operations $[x := 0]$, $[x > 5, x := 0]$, $[x < 1]$ and $[x = 3, x := 0]$, where $[x > 5, x := 0]$, for instance, means that x is tested against 5 and it is reset simultaneously. In the second row, all the operations on x are represented by means of clocks x_0 and x_1. Based on the value of x_v at i, the active clock at that position is used to realize the corresponding operation. A continuous line identifies the regions where either clock x_0 or x_1 is active. The third row shows the constraints on σ that are enforced by the operations on x.

4.2 Encoding the network ($\varphi_{\mathcal{N}}$).

Formula $\varphi_{\mathcal{N}}$ encodes both the relation $\xrightarrow{e}$ between pairs of configurations and all (and only) the conditions of Def. 3.3 defining a trace. However, $\varphi_{\mathcal{N}}$ does not express any restriction on automata synchronization and it does not impose any specific liveness condition. The discrete positions of the CLTLoc model render the configurations of the network evolving over the continuous time by means of a discrete sequence of points. All those positions in the model represent the discrete transitions performed by the automata of $\mathcal{N}$ that modify the values of variables, clocks and locations and also the time transitions that produce the elapsing of time. The model of the formula $\varphi_{\mathcal{N}}$ is thus a (representation of a) possible trace realized by the network $\mathcal{N}$.

A generic configuration $(\mathtt{l}, v_{\mathrm{var}}, v)$ of $\mathcal{N}$ is represented in the CLTLoc formula by means of the values of clocks and variables and a set of auxiliary variables representing locations and transitions of the automata. An array $\mathtt{l}$ of K integer variables in the CLTLoc formula encodes the location of each automaton in the network, with $\mathtt{l}[k] \in \{0, \dots, |Q_k| - 1\}$ for every $1 \leq k \leq K$. Given an enumeration of the elements in Q_k, $\mathtt{l}[k] = i$ indicates that $\mathcal{A}_k$ is in the location q_i of Q_k. An array $\mathtt{t}$ of K integer variables encodes the transitions of each automaton in the network, with $\mathtt{t}[k] \in \{0, \dots, |T_k| - 1\} \cup \{\natural\}$, for every $1 \leq k \leq K$. Given an enumeration of the elements in T_k, $\mathtt{t}[k] = i$ indicates the execution of transition t_i of T_k, while $\mathtt{t}[k] = \natural$ indicates that no transition of T_k is performed (it represents the symbol _ in the discrete transitions of traces). An array $\mathrm{edge}^{](}$ of K Booleans represents the kind of transition taken by each automaton $\mathcal{A}_k$. If $\mathrm{edge}^{](}[k]$ is true, then the configuration change of $\mathcal{A}_k$ at the current position is closed-open; otherwise, it is open-closed. Finally, for each variable $n \in Int$, a corresponding CLTLoc integer variable is introduced.

The configuration change determined by a discrete transition is explicitly encoded with a formula that expresses the effect of the transition on the network configuration. Conversely, since in CLTLoc

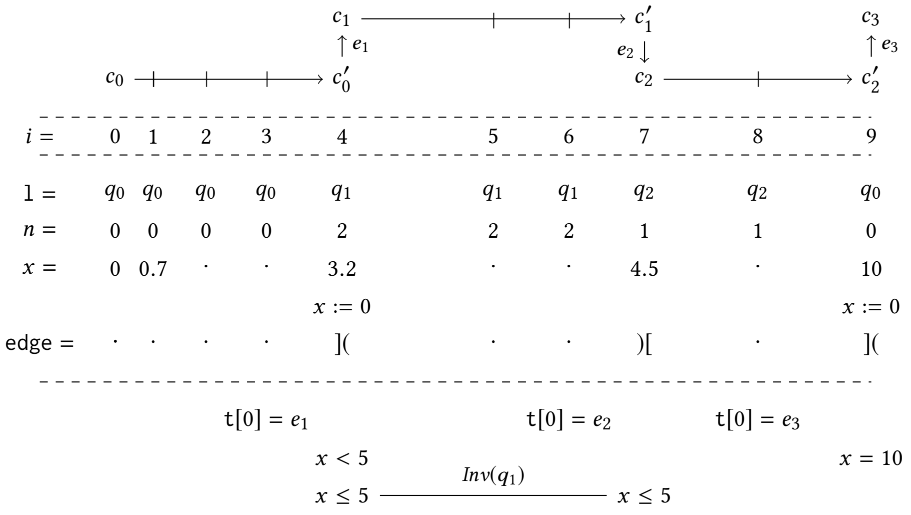


Fig. 5. Interpretation of atom in $\varphi_{\mathcal{N}}$.

time progress is inherent in the model, the encoding does not explicitly deal with time transitions of traces because between any pair of adjacent positions i and $i+1$ the time always advances. To facilitate understanding and future discussions, a trace is written as:

$$(\mathtt{l}_0, v_{\text{var},0}, v_0) \xrightarrow{\delta_0\Lambda_0} (\mathtt{l}_1, v_{\text{var},1}, v_1) \xrightarrow{\delta_1\Lambda_1} (\mathtt{l}_2, v_{\text{var},2}, v_2) \dots$$

where time and discrete transitions are paired together and the notation $(\mathtt{l}_i, v_{\text{var},i}, v_i) \xrightarrow{\delta_i\Lambda_i} (\mathtt{l}_{i+1}, v_{\text{var},i+1}, v_{i+1})$ is simply a rewriting of

$$(\mathtt{l}_i, v_{\text{var},i}, v_i) \xrightarrow{\delta_i} (\mathtt{l}'_i, v'_{\text{var},i}, v'_i) \xrightarrow{\Lambda_i} (\mathtt{l}_{i+1}, v_{\text{var},i+1}, v_{i+1}).$$

for some configuration $(\mathtt{l}'_i, v'_{\text{var},i}, v'_i)$. Formula $\varphi_{\mathcal{N}}$ is built by assuming that the configuration of the network does not change over the intervals of time delimited by a pair of positions of the CLTLoc model, except for clocks progressing. Hence, any pair of positions i and $i+1$ of the model of $\varphi_{\mathcal{N}}$ represents (the pair of transitions) $(\mathtt{l}_i, v_{\text{var},i}, v_i) \xrightarrow{\delta_i\Lambda_i} (\mathtt{l}_{i+1}, v_{\text{var},i+1}, v_{i+1})$.

The atomic propositions and variables, appearing in $\varphi_{\mathcal{N}}$, are interpreted with the following meaning:

- if $l[k] = j$ holds at position i, then automaton $\mathcal{A}_k$ is in state q^k_j over the interval of time that starts at i and ends in $i+1$.
- if $\mathtt{t}[k] = j$ holds at position i, then automaton $\mathcal{A}_k$ performs transition j in $i+1$.

Example 4.2. Figure 5 shows a trace of the automaton depicted in Fig. 2(b) that consists of various time transitions and three discrete transitions associated with events e_1, e_2 and e_3. To facilitate readability, the discrete transitions such that $\Lambda[0] = \_$ are indicated with a vertical bar and the discrete transitions where at least one automaton executes (in the next position) a transition are drawn by showing the primed configurations. Every discrete transition corresponds to a unique position in the CLTLoc model and every time transition determines the time progress between pairs of adjacent positions. The first area below the trace shows the discrete positions i of the CLTLoc

Table 4. Encoding of the automaton.

$\varphi_1 := \bigwedge_{k\in[1,K]} (\mathtt{l}[k]=0)$	$\varphi_2 := \bigwedge_{n\in Int} n = v^0_{\mathrm{var}}(n)$	$\varphi_3 := \bigwedge_{k\in[1,K]} Inv(\mathtt{l}[k])$
$\varphi_4 := \bigwedge_{\substack{k\in[1,K]\\ q\in Q_k}} ((\mathtt{l}[k]=q \wedge \mathtt{t}[k]=\natural) \rightarrow \mathcal{X}(Inv(q) \wedge r_1(Inv(q)))$		
$\varphi_5 := \bigwedge_{k\in[1,K], t\in T_k} \mathtt{t}[k]=t \rightarrow \left(\mathtt{l}[k]=t^- \wedge \phi_\xi \wedge \mathcal{X}(\mathtt{l}[k]=t^+ \wedge \phi_\gamma \wedge \phi_\mu \wedge \phi_\zeta \wedge \phi_{edge}(t^-,t^+,k)\right)$ $\phi_{edge}(a,b,i) := \phi_{\alpha](}(a,b,i) \vee \phi_{\alpha)[}(a,b,i)$ $\phi_{\alpha](}(a,b,i) := Inv(a) \wedge r_2(Inv_w(b)) \wedge \mathrm{edge}^{](}[i]$ $\phi_{\alpha)[}(a,b,i) := Inv_w(a) \wedge r_2(Inv(b)) \wedge \neg\mathrm{edge}^{](}[i]$		
$\varphi_6 := \bigwedge_{k\in[1,K], q,q'\in Q_k \mid q\neq q'} \left(((\mathtt{l}[k]=q) \wedge \mathcal{X}(\mathtt{l}[k]=q')) \rightarrow \bigvee_{t\in T_k, t^-=q, t^+=q'} (\mathtt{t}[k]=t)\right)$		
$\varphi_7 := \bigwedge_{x\in X} \left(\mathcal{X}(x_0=0 \vee x_1=0) \rightarrow \bigvee_{\substack{k\in[1,K]\\ t\in T_k \mid x\in t_s}} \mathtt{t}[k]=t\right)$	$\varphi_8 := \bigwedge_{n\in Int} \left((\neg(n=\mathcal{X}(n))) \rightarrow \bigvee_{\substack{k\in[1,K]\\ t\in T_k \mid n\in U(t)}} \mathtt{t}[k]=t\right)$	

model and the second one, for each position i, provides the values of the variables representing location $\mathtt{l}$, variable n, clock x (a dot stands for a monotonically increasing positive value), and the value of variable edge (a dot represents an irrelevant assignment to edge). The first discrete transition labeled with e_1 occurs at position 4, where the guard $x < 5$ holds; at that moment, clock x, whose value is equal to 3.2, is reset and the location changes from q_0 to q_1. The second transition—associated with event e_2—occurs at position 7 when $x = 4.5$ holds, before the value of x violates the invariant $x \leq 5$ of location q_1, and produces the change of location from q_1 to q_2. The last transition, associated with event e_3, occurs at position 9 with $x = 10$, it resets x and changes location to q_0. In the CLTLoc model, discrete transitions are represented one position earlier than the position where they actually occur, namely at position 3, 6, and 8, respectively.

Remark 5. *As specified in Definition 3.3, Condition (3), the first configuration change is associated with a time transition. Thus, it is not possible to fire a discrete transition at position* $i = 0$.

The third segment of Fig. 5 shows the exact positions where transitions $\mathtt{t}[0] = e_1$, $\mathtt{t}[0] = e_2$ and $\mathtt{t}[0] = e_3$ occur (first line), the positions where the guards are evaluated in the CLTLoc model (second line) and the sequence of positions where the invariant of q_1 holds (third line). For convenience of notation, the assignment for edge is shown by means of symbols](and)[. At positions 4, 7 and 9 it is shown one among the possible assignments that are compatible with the clock values in the model. For instance,)[is also possible at position 4.

A network of TA is transformed into a CLTLoc formula using the formulae in the Table 4. In the following, the invariant of location q is indicated with $Inv(q)$ and the weak version of $Inv(q)$, where all relations $<, >$ are replaced with $\leq, \geq$ and the equalities are replaced with false, is denoted with $Inv_w(q)$. With slight abuse of notation, $Inv(q)$ and $Inv_w(q)$ are used in Fig. 4 to indicate the CLTLoc formula corresponding to the invariant of q and its weak version.

Before explaining the formulae of Table 4 in details, a short description is first provided. Formulae φ_1, φ_2 and φ_3 specify the initial conditions that must hold in the TA. Formula φ_4 specifies the behavior of a TA when a time transition is fired. Formula φ_5 and formulae φ_6-φ_8 define, respectively, the necessary and the sufficient conditions that must hold when a discrete transition is performed with a symbol different from $\natural$, i.e., when the corresponding symbol in the trace is not _[4]. More precisely, formulae φ_6-φ_8 force the execution of (at least) a transition in the network if a reset or a change of the value of variables or locations occurs and they prevent a variation in the values of the model that is not caused by the occurrence of a transition. Each of the formulae is discussed in detail in the following.

- Formula φ_1 specifies that at position 0 every automaton is in its initial state;
- Formula φ_2 specifies that at position 0 every variable n is assigned its initial value $v_{\mathrm{var},0}(n)$;
- Formula φ_3 specifies that at position 0 the invariant of the initial state of each TA holds;
- Formula φ_4 encodes the case (1)b of Def. 3.2 requiring that, for every $1 \le k \le K$, if automaton $\mathcal{A}_k$ does not perform a transition when other TA do, then the clocks still satisfy the invariant of the current location $\mathtt{l}[k]$. Formula $Inv(q) \wedge r_1(Inv(q))$ guarantees that the values of the active clocks satisfy $Inv(q)$, even in the case when they are reset. Unlike $Inv(q)$, $r_1(Inv(q))$ does not make use of formulae $\phi_{x\sim d}$ to evaluate constraints $x \sim d$ when the clock assignments are equal to 0. In fact, the evaluation of a constraint $x \sim d$ through $\phi_{x\sim d}$ only depends on the active value of x, which is always different from 0 by definition (see Sec. 4.1). To this goal, the mapping r_1, defined below, replaces every constraint of the form $x \sim d$ with the value of $0 \sim d$ if x is reset. Given an atomic formula $\beta(x)$ of the form $x \sim d$, where $\sim \in \{<, \le, =, \ge, >\}$, let $\beta_{[x\leftarrow c]}$ be true or false depending on the value of the formula obtained by replacing x in $\beta(x)$ with the constant c. Then, for a clock constraint γ, let $r_1(\gamma)$ be defined as the formula obtained from γ by replacing, for all clocks x, each occurrence of an atomic formula $\beta(x)$ with the formula:

$$(x_0 = 0 \vee x_1 = 0) \rightarrow \beta_{[x\leftarrow 0]}.$$

  When x is reset, x_0 or x_1 are equal to 0, and $\beta_{[x\leftarrow 0]}$ establishes the value of $\beta(x)$. Since a CLTLoc model represents the occurrence of a transition one time position before the effect, φ_4 imposes that the invariant $Inv(q)$ associated with the current location $\mathtt{l}[k] = q$ is satisfied in the next position if no transition is taken in the current one, i.e, when $t[k] = \natural$. The value of $\mathrm{edge}^{\rceil(}[k]$ is irrelevant because no transition of $\mathcal{A}_k$ is occurring. Hence, no constraint is specified for it.
- Formula φ_5 encodes the case (1)a of Def. 3.2. Similarly to formula φ_4, the value of active clocks and their resets have to be considered carefully in the evaluation of the invariants of $\mathtt{l}[k]$ and $\mathtt{l}'[k]$. In the definition of discrete transitions of Def. 3.2, the invariant of $\mathtt{l}[k]$ is always evaluated with respect to v whereas the one of $\mathtt{l}'[k]$ is evaluated with respect to v'. Function r_2 is used to encode the conditions in (1)(a)v and (1)(a)iv, that require the satisfaction of $Inv(\mathtt{l}'[k])$, or possibly its weak version, with v'. Let γ, β and $\beta_{[x\leftarrow c]}$ be formulae defined as above and let $r_2(\gamma)$ be the formula where all the occurrences $x \sim d$ are replaced with the formula

$$((x_0 > 0 \wedge x_1 > 0) \rightarrow \phi_{x\sim d}) \wedge ((x_0 = 0 \vee x_1 = 0) \rightarrow (x \sim d)_{[x\leftarrow 0]}).$$

  For instance, by means of r_2, the value of a constraint $x \sim d$ occurring in $Inv(\mathtt{l}'[k])$ is either $(x \sim d)_{[x\leftarrow 0]}$, if x is reset by the transition (i.e., $v'(x) = 0$); or $\phi_{x\sim d}$ if it is not. In the latter case, its value is determined by the active clock for x that is equal to $v'(x)$.

[4]In $\varphi_5 \ldots \varphi_8$, symbols of Act_τ do not appear, as events are only used to define how the TA synchronize and they will be discussed in the next section.

Let t be $q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q' \in T_k$, let ϕ_γ be the CLTLoc formula expressing the guard γ, let ϕ_ξ be the formula expressing the constraints on the integer variables, let ϕ_ζ be the formula $\bigwedge_{x\in\zeta}(x_0 = 0 \vee x_1 = 0)$ encoding in CLTLoc the effect of resets applied by t on clocks in ζ and let ϕ_μ be the formula translating the assignments of the form $n := exp$ that appear in t into a (semantically) equivalent CLTLoc formula.
Recall that $\mathtt{t}[k] = t$ represents the execution of transition t in the next position of time and that t^+, t^- are the locations q and q' connected by t. If $\mathtt{t}[k] = t$ holds at position i (hence, transition t is actually performed by $\mathcal{A}_k$ at $i + 1$) then:

(1) Automaton $\mathcal{A}_k$ is currently in q and changes location to q' in the next position of time. Hence, $\mathtt{l}[k]$ at i and $i + 1$ is, respectively, $q = t^-$ and $q' = t^+$.
(2) The condition on the integer variables holds. Formula ϕ_ξ is satisfied at position i because the effect of t on the integer variables is enforced at $i + 1$.
(3) The condition on the clocks holds. Formula ϕ_γ is satisfied by the clock assignments in $i + 1$.
(4) All the clock resets and variable assignments are performed. Hence, formulae ϕ_ζ and ϕ_μ hold at position $i + 1$, i.e., resets and updates are performed when t is actually taken.
(5) The configuration change is either open-closed or closed-open. Formula ϕ_{edge} encodes the cases (1)(a)v and (1)(a)iv. If the configuration change is closed-open then, according to (1)(a)v, $v \models Inv(\mathtt{l}[k])$ and $v' \models_w Inv(\mathtt{l}'[k])$ must hold. The first condition is ensured by $Inv(t^-)$, while the second by $r_2(Inv_w(t^+))$, where $t^- = \mathtt{l}[k]$ and $t^+ = \mathtt{l}'[k]$. Since the transition is closed-open, then $\mathtt{edge}^{](}[i]$ is set to true. If the configuration change is open-closed then, according to (1)(a)iv, $v \models_w Inv(\mathtt{l}[k])$ and $v' \models Inv(\mathtt{l}'[k])$ must hold. The first condition is ensured by $Inv_w(t^-)$, while the second by $r_2(Inv(t^+))$, where $t^- = \mathtt{l}[k]$ and $t^+ = \mathtt{l}'[k]$. Since the transition is not closed-open, then $\mathtt{edge}^{](}[i]$ is false.

- Formula φ_6 specifies that if automaton $\mathcal{A}_k$ modifies its location from q to q' over two adjacent positions, then a transition $t \in T_k$ such that $q = t^-$ and $q' = t^+$ is taken at $i + 1$—i.e., $\mathtt{t}[k] = t$ holds at position i.
- Formula φ_7 specifies that if a reset of x (i.e., either $x_0 = 0$ or $x_1 = 0$) occurs at $i + 1$ then a transition resetting clock x is performed at $i + 1$—i.e., $\mathtt{t}[k] = t$ holds at position i, for some $1 \leq k \leq K$ and $t \in T_k$ such that $x \in t_s$ (where t_s is the set of clocks that transition t resets).
- Formula φ_8 specifies that if the value of variable n in $i + 1$ is not equivalent to the one in i then a transition modifying n is performed at $i + 1$—i.e., $\mathtt{t}[k] = t$ holds at position i, for some $1 \leq k \leq K$ and $t \in T_k$ such that $n \in U(t)$ (where $U(t)$ is the set of integer variables that transition t updates by means of the assignments in μ).

Formula $\varphi_\mathcal{N}$ encoding the network is then defined in Formula (1).

$$\varphi_\mathcal{N} = \phi_{clock} \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \mathcal{G}\Big(\bigwedge_{4\leq i\leq 8} \varphi_i\Big) \tag{1}$$

Remark 6. *The proposed encoding allows the simultaneous execution, by different automata, of transitions with edges of type $\alpha^{)[}$ and $\alpha^{](}$.*

The correctness of the CLTLoc encoding is demonstrated by showing a correspondence between the traces of a network $\mathcal{N}$ (Def 3.3) and the models (π, σ, ι) of the CLTLoc formula $\varphi_\mathcal{N}$. Without loss of generality, assume that the set of clocks X of $\mathcal{N}$ is not empty (if $X = \emptyset$, one could always add a clock that is never reset, nor tested, and the behavior of the network would not change).

At the core of the proof there is a mapping, ρ, between traces of TA and CLTLoc models. First of all, every trace η of TA can be given the form

$$(\mathtt{l}_0, v_{\text{var},0}, v_0) \xrightarrow{\delta_0\Lambda_0} (\mathtt{l}_1, v_{\text{var},1}, v_1) \xrightarrow{\delta_1\Lambda_1} (\mathtt{l}_2, v_{\text{var},2}, v_2) \ldots$$

since any pair of consecutive time transitions can be seen as a pair of time transitions separated by a discrete transition in which the action is _ (i.e., nothing happens in between). Then, given a trace η, a CLTLoc model (π, σ, ι) that belongs to $\rho(\eta)$ can be built as follows.

For every position $h \in \mathbb{N}$, the function ι assigns each CLTLoc variable $\mathtt{l}[k]$ to location $\mathtt{l}_h[k]$ (that is, $\iota(h, \mathtt{l}[k]) = \mathtt{l}_h[k]$). The value of variables $n \in Int$ is defined as $\iota(h, n) = v_{\text{var},h}(n)$. Function π assigns values to the atomic propositions of $\mathtt{l}_h$, for every index $h \in \mathbb{N}$: $p \in \pi(h)$ if, and only if, $p \in L(\mathtt{l}_h[k])$, for some k. The clock valuation σ specifies the assignments to both active and inactive clocks. Recall that $(\mathtt{l}_h, v_{\text{var},h}, v_h) \xrightarrow{\delta_h \Lambda_h} (\mathtt{l}_{h+1}, v_{\text{var},h+1}, v_{h+1})$ is a shortcut for

$$(\mathtt{l}_h, v_{\text{var},h}, v_h) \xrightarrow{\delta_h} (\mathtt{l}'_h, v'_{\text{var},h}, v'_h) \xrightarrow{\Lambda_h} (\mathtt{l}_{h+1}, v_{\text{var},h+1}, v_{h+1})$$

where the time transition $(\mathtt{l}_h, v_{\text{var},h}, v_h) \xrightarrow{\delta_h} (\mathtt{l}'_h, v'_{\text{var},h}, v'_h)$ only updates clocks, while locations and integer variables are unchanged. For convenience of writing, let x_a and x_i be, respectively, the *active* and the *inactive* clocks associated with x in a given position. Initially, the active clock is x_0 (i.e., x_a is x_0), and its value is 0; that is, $\sigma(0, x_0) = 0$ and $\iota(0, \mathrm{x}_v) = 0$ (the value of x_1—i.e., x_i—is arbitrary, hence it can be any positive value). For all $h \in \mathbb{N}$, $x \in X$, define $\sigma(h+1, x_a) = \sigma(h, x_a) + \delta_h$; and also $\sigma(h+1, x_i) = \sigma(h, x_i) + \delta_h$ unless x is reset in position $h+1$ of the trace, in which case $\sigma(h+1, x_i) = 0$; clock x_i becomes the active clock from $h+1$ (excluded), and the value of $\iota(h+2, \mathrm{x}_v) = (\iota(h+1, \mathrm{x}_v) + 1) \mod 2$.

The value of predicate $\mathtt{t}[k]$ at position h is defined based on configurations $(\mathtt{l}'_h, v'_{\text{var},h}, v'_h)$, $(\mathtt{l}_{h+1}, v_{\text{var},h+1}, v_{h+1})$ and on symbol $\Lambda_h[k]$. In particular, define $\iota(h, \mathtt{t}[k]) = \natural$ when $\Lambda_h[k] = \_$. Instead, define $\iota(h, \mathtt{t}[k]) = t$ when there is $t = q \xrightarrow{\gamma, \xi, \alpha, \zeta, \mu} q'$ such that $\mathtt{l}'_h[k] = q$, $\mathtt{l}_{h+1}[k] = q'$, $\Lambda_h[k] = \alpha$, and v'_h, v_{h+1}, $v'_{\text{var},h}$ $v_{\text{var},h+1}$ are compatible with γ, ξ, ζ, μ according to the semantics of Def. 3.2. Finally, the value of $\mathrm{edge}^{]\!(}[k]$ is set according to $\Lambda_h[k]$: if $\Lambda_h[k] = (\alpha, b)$ for some action α, then $\mathrm{edge}^{]\!(}[k] \in \pi(h+1)$ if, and only if, $b = \mathtt{ie}$; otherwise, if $\Lambda_h[k] = \_$, the value of $\mathrm{edge}^{]\!(}[k]$ in $h+1$ is arbitrary (it is also arbitrary in the origin).

Notice that, given a trace η, its mapping $\rho(\eta)$ contains more than one CLTLoc model (for example, the value of x_i in the origin is arbitrary). The inverse mapping ρ^{-1}, instead, defines, for each CLTLoc model (π, σ, ι), a unique trace $\eta = \rho^{-1}((\pi, \sigma, \iota))$; indeed, the presence of at least a clock x in $\mathcal{N}$ entails that at each position $h+1$ at least one of the two corresponding CLTLoc clocks x_0, x_1 is not reset, which in turn uniquely identifies the delay δ_h.

The rest of this section sketches the proof for the following result.

Theorem 4.3. *Let $\mathcal{N}$ be a network of TA and $\varphi_{\mathcal{N}}$ be its corresponding CLTLoc formula. For every trace η of $\mathcal{N}$, every CLTLoc model (π, σ, ι) such that $(\pi, \sigma, \iota) \in \rho(\eta)$ is a model of $\varphi_{\mathcal{N}}$. Conversely, for all CLTLoc models (π, σ, ι) of $\varphi_{\mathcal{N}}$, $\rho^{-1}((\pi, \sigma, \iota))$ is a trace of $\mathcal{N}$.*

Proof. *From traces to models.*

Let $\eta = (\mathtt{l}_0, v_{\text{var},0}, v_0) \xrightarrow{\delta_0 \Lambda_0} (\mathtt{l}_1, v_{\text{var},1}, v_1) \xrightarrow{\delta_1 \Lambda_1} (\mathtt{l}_2, v_{\text{var},2}, v_2) \dots$ be a trace of $\mathcal{N}$, and $(\pi, \sigma, \iota) \in \rho(\eta)$. Formulae φ_1, φ_2 and φ_3 of $\varphi_{\mathcal{N}}$ are satisfied since they trivially hold at position 0.

The following arguments show that, at every position $h \in \mathbb{N}$, CLTLoc formulae φ_4, φ_5 are satisfied by (π, σ, ι) (i.e., $(\pi, \sigma, \iota), h \models \varphi_4$ and $(\pi, \sigma, \iota), h \models \varphi_5$ both hold). Different cases are considered, depending on the nature of $\Lambda_h[k]$ (where $1 \le k \le K$).

(1) Case $\Lambda_h[k] = \_$. In this case, the conditions (1)b of Definition 3.2 hold in the trace:
   - $\mathtt{l}_h[k] = \mathtt{l}_{h+1}[k]$;
   - $v'_h \models Inv(\mathtt{l}_h[k])$ and $v_{h+1} \models Inv(\mathtt{l}_{h+1}[k])$.

   The antecedent of φ_4 holds at position h and formula $Inv(\mathtt{l}[k]) \wedge r_1(Inv(\mathtt{l}[k]))$ holds at $h+1$—hence satisfying the entailment—because the second condition of (1)b holds in the trace.

Indeed, all clock constraints β in $Inv(\mathtt{l}[k])$ of the form $x \sim d$ correspond to formula ϕ_β in φ_4, evaluated with the values of the active clocks at position $h+1$. Hence, $v'_h(x)$ satisfies β if, and only if, ϕ_β holds at position $h+1$, since by construction $\sigma(h+1, x_a) = v'_h(x)$. Moreover, in case of reset of x at $h+1$ in the trace (in which case, by construction, $\sigma(h+1, x_i) = v_{h+1}(x) = 0$ holds), every clock constraint β of the form $x \sim d$ corresponds to CLTLoc formula $r_1(\beta)$, which in turn reduces to the constant $\beta_{[x \leftarrow 0]}$, and which equals to the evaluation of β in $v_{h+1}(x)$.
If $\Lambda_h[k] = \_$, then by construction $\iota(h, \mathtt{t}[k]) = \natural$ and φ_5 is vacuously satisfied.

(2) Case $\Lambda_h[k] = \alpha^{](}$. Let t be the transition such that $\iota(h, \mathtt{t}[k]) = t$. By Def. 3.2 condition (1)a , the constraints hold in the trace (notice that $v_{\mathrm{var},h} = v'_{\mathrm{var},h}$):
- $v'_h \models \gamma$ and $v'_{\mathrm{var},h} \models \xi$,
- $v_{h+1}(x) = 0$ for all $x \in \zeta$,
- $(v_{\mathrm{var},h+1}, v'_{\mathrm{var},h}) \models \mu$,
- $v'_h \models Inv(\mathtt{l}_h[k])$ and
- $v_{h+1} \models_w Inv(\mathtt{l}_{h+1}[k])$.

Thus, formula φ_4 vacuously holds in h.
Formula φ_5 holds since the antecedent of φ_5 holds for the case $\mathtt{t}[k] = t$ and the consequent is satisfied as follows. By construction, it holds that $\iota(h, \mathtt{l}[k]) = \mathtt{l}_h[k]$ and $\iota(h+1, \mathtt{l}[k]) = \mathtt{l}_{h+1}[k]$. Formula ϕ_γ holds at position $h+1$ with $\sigma(h+1, x_a) = v'_h(x)$, for every clock x; also, ϕ_ξ holds at position h with $\iota(h, n) = v_{\mathrm{var},h}(n) = v'_{\mathrm{var},h}(n)$ for every variable n. Formulae ϕ_ζ and ϕ_μ hold at $h+1$, since $\sigma(h+1, x_i) = v_{h+1}(x) = 0$ for all clock $x \in \zeta$, and $\iota(h+1, n) = v_{\mathrm{var},h+1}(n)$ for all variable $n \in Int$. The first condition on the invariants of $\mathtt{l}[k]$ is evaluated as follows. Each clock constraint β in $Inv(t^-)$ corresponds to CLTLoc formula ϕ_β, evaluated with the values of the active clocks at position $h+1$ of σ. Hence, since $\sigma(h+1, x_a) = v'_h(x)$, ϕ_β holds at position $h+1$ if, and only if, $v'_h(x)$ satisfies β. The formula $r_2(Inv_w(t^+))$ must hold at $h+1$ to guarantee the enforcement of the last condition in the model. By definition of r_2, each clock constraint β in $Inv_w(t^+)$ of the form $x \sim d$ corresponds to formula ϕ_β, evaluated with the values in σ of the active clocks at position $h+1$, if the clock is not reset; otherwise, β reduces to $\beta_{[x \leftarrow 0]}$, whose value is that of β when valuation v_{h+1} is considered. By construction, $\mathrm{edge}^{](}[k] \in \pi(h+1)$ holds, so $\phi_{edge}(t^-, t^+, k)$ also holds in $h+1$.

(3) Case $\Lambda_h[k] = \alpha^{)[}$. The proof is similar to the previous one. The only differences are the conditions on the invariants, that are $Inv_w(t^-)$ and $r_2(Inv(t^+))$. However, the same arguments of the previous case hold.

Formulae φ_6 and φ_8 are trivially satisfied when the location $\mathtt{l}[k]$ does not change in η—hence it does not change in (π, σ, ι)—and for all variables n that have the same value in h and $h+1$; similarly, φ_7 is satisfied when a clock x is not reset in $h+1$ (hence, both x_a and x_i are not 0). Otherwise, if between positions h and $h+1$ in the trace the location changes, or a variable is updated, or a clock is reset, then there must be a transition taken in the TA, hence by construction in (π, σ, ι) both the antecedents and the consequences of formulae $\varphi_6 - \varphi_8$ trivially hold.

From models to traces. Let (π, σ, ι) be a model of $\varphi_{\mathcal{N}}$. The proof shows that $\eta = \rho^{-1}((\pi, \sigma, \iota)) = (\mathtt{l}_0, v_{\mathrm{var},0}, v_0) \xrightarrow{\delta_0 \Lambda_0} (\mathtt{l}_1, v_{\mathrm{var},1}, v_1) \xrightarrow{\delta_1 \Lambda_1} (\mathtt{l}_2, v_{\mathrm{var},2}, v_2) \ldots$ is a trace of $\mathcal{N}$ according to Def. 3.3 (and the related Def. 3.2).

Since formulae $\varphi_1, \varphi_2, \varphi_3$ hold in at position 0 of (π, σ, ι), the values of $\mathtt{l}_0, v_{\mathrm{var},0}, v_0$ defined by mapping ρ^{-1} constitute an initial configuration of $\mathcal{N}$ according to Def. 3.3.

The following arguments show that, at each position $h \in \mathbb{N}$ of η, for all k (with $1 \le k \le K$) the conditions of Def. 3.2 hold. Separate cases are considered, depending on the value of $\iota(h, \mathtt{t}[k])$.

(1) $\iota(h, \mathrm{t}[k]) = \natural$. Then, $\mathrm{l}[k]$ is the same in h and $h+1$ and φ_4 ensures that $Inv(\mathrm{l}[k]) \wedge r_1(Inv(\mathrm{l}[k]))$ holds at $h+1$, which in turn entails that $v'_h \models Inv(\mathrm{l}[k])$ and $v_{h+1} \models Inv(\mathrm{l}[k])$ both hold. Hence, condition (1)b holds.

(2) $\iota(h, \mathrm{t}[k]) = t \neq \natural$ and $\mathrm{edge}^{](}[k] \in \pi(h+1)$, where $t = q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ is a transition of $\mathcal{A}_k$. Since φ_5 holds in h, then the following conditions hold:
- $\iota(h, \mathrm{l}[k]) = q$ and $\iota(h+1, \mathrm{l}[k]) = q'$,
- ϕ_ξ holds in h and
- ϕ_γ, ϕ_μ, ϕ_ζ and $\phi_{\mathrm{edge}}(q, q', k)$ hold in $h+1$.

All conditions in (1)a, and in particular those in (1)(a)v corresponding to the case $\alpha^{](}$, for some action α, hold as follows:

(a) $v'_h \models \gamma$ and $v'_{\mathrm{var},h} \models \xi$ (condition (1)(a)i) are guaranteed by ϕ_γ holding in $h+1$ and ϕ_ξ holding in h, respectively (notice that in $h+1$ variables are updated by μ and clocks are possibly reset, but γ is evaluated through the active clocks).
(b) $v_{h+1}(x) = 0$ (condition (1)(a)ii) holds, for all $x \in \zeta$, since ϕ_ζ holds in $h+1$.
(c) $(v_{\mathrm{var},h+1}, v_{\mathrm{var},h}) \models \mu$ (condition (1)(a)iii) holds, since ϕ_μ holds in $h+1$.
(d) $v'_h \models Inv(\mathrm{l}'_h[k])$ holds because, by $\phi_{\mathrm{edge}}(q, q', k)$, $Inv(q)$ holds in $h+1$, and ρ^{-1} defines that $\mathrm{l}'_h[k]$ is q; then, the first condition of (1)(a)v holds.
(e) $v_{h+1} \models_w Inv(\mathrm{l}_{h+1}[k])$ holds because, again by $\phi_{\mathrm{edge}}(q, q', k)$, $r_2(Inv_w(q'))$ holds in $h+1$, and ρ^{-1} defines that $\mathrm{l}_{h+1}[k]$ is q'; then, the second condition of (1)(a)v holds.

(3) $\iota(h, \mathrm{t}[k]) = t \neq \natural$ and $\mathrm{edge}^{](}[k] \notin \pi(h+1)$. The proof, which now focuses on condition (1)(a)iv, is similar to the previous case—with the only differences being the conditions on the invariants, which are now $Inv_w(q)$ and $r_2(Inv(q'))$—and it is omitted for simplicity.

Condition (1)c follows from φ_7 and φ_8 holding at h, which impose the occurrence of a transition that resets a clock x or modifies the value of a variable n if x is reset or n is updated at position $h+1$. Finally, condition (2) holds because, as mentioned above, formula φ_4 imposes that $Inv(\mathrm{l}[k])$ holds at $h+1$, hence $v'_h \models Inv(\mathrm{l}[k])$ also holds, which in turn implies that $v'_h \models_w Inv(\mathrm{l}[k])$ holds (notice that a model for $\varphi_\mathcal{N}$ cannot reach a location that includes constraints of the form $x = d$ in its invariant, since time is strictly monotonic, and the residence time in the location cannot be null). In addition, formula φ_5 defines that $Inv(\mathrm{l}[k])$ or $Inv_w(\mathrm{l}[k])$ holds at $h+1$, and both entail that $v'_h \models_w Inv(\mathrm{l}[k])$ holds. □

4.3 Encoding liveness, synchronization and edge constraints (φ_l, φ_s and φ_{ef})

As seen in Section 3.2, different liveness conditions and synchronization mechanisms for networks of TA can be considered. This section describes how the liveness conditions and synchronization mechanisms presented in Section 3.2 can be encoded in CLTLoc. Several liveness conditions could be required for a network of TA, so Formula φ_l captures a conjunction of the following conditions, each one encoded by a CLTLoc formula in Table 5 (if no liveness condition is required, φ_l reduces to true).

- *Strong transition liveness:* at any time instant, at least one transition in *every* automaton is eventually fired.
- *Weak transition liveness:* at any time instant, at least one transition in *at least one* automaton is eventually fired.
- *Strong guard liveness:* at any time instant, *every* automaton eventually reaches a state that has an outgoing transition whose guard holds.
- *Weak guard liveness:* at any time instant, *at least one* automaton eventually reaches a state that has an outgoing transition whose guard holds.

Table 5. Formulae encoding the different liveness conditions.

Liveness	**Property**
Strong transition	$\bigwedge_{k\in[1,K]} \mathcal{G}\left(\mathcal{F}\left(\bigvee_{t\in T_k} \mathtt{t}[k]=t\right)\right)$
Weak transition	$\mathcal{G}\left(\mathcal{F}\left(\bigvee_{k\in[1,K],\, t\in T_k} \mathtt{t}[k]=t\right)\right)$
Strong guard	$\mathcal{G}\left(\bigwedge_{k\in[1,K]}\left(\bigwedge_{q\in Q_k} \mathtt{l}[k]=q \rightarrow \mathcal{F}\left(\bigvee_{t\in T_k, t^-=q} \phi_{t_\gamma} \wedge \phi_{t_{\gamma var}}\right)\right)\right)$
Weak guard	$\mathcal{G}\left(\bigvee_{k\in[1,K]}\left(\bigwedge_{q\in Q_k} \mathtt{l}[k]=q \rightarrow \mathcal{F}\left(\bigvee_{t\in T_k, t^-=q} \phi_{t_\gamma} \wedge \phi_{t_{\gamma var}}\right)\right)\right)$

Synchronization is encoded by relying on the following abbreviations, where $\alpha \in Act_\tau$, $k, h \in [1, K]$ and S is a set of indices in $[1, K]$. Recall that, given a transition t, $t_e \in Act_\tau$ represents the symbols that labels t.

$$\phi_{\text{sync-on}}(k,\alpha) := \bigvee_{t\in T_k \mid t_e=\alpha} (\mathtt{t}[k]=t) \tag{2}$$

$$\phi_{\text{sync-on-but}}(S,\alpha) := \bigvee_{g\in\{i \mid i\in[1,K]\}\setminus S} \phi_{\text{sync-on}}(g,\alpha) \tag{3}$$

$$\phi_{\text{same-edge}}(k,h) := \mathcal{X}(\mathtt{edge}^{](}[k] \leftrightarrow \mathtt{edge}^{](}[h]) \tag{4}$$

Formula $\phi_{\text{sync-on}}(k,\alpha)$ specifies that a transition t of $\mathcal{A}_k$ labeled with the action α is fired. Formula $\phi_{\text{sync-on-but}}(S,\alpha)$ specifies that a transition t labeled with the action α, and belonging to a TA whose index does not belong to set S, is fired. Finally, $\phi_{\text{same-edge}}(k,h)$ specifies that the transitions taken by $\mathcal{A}_k$ and $\mathcal{A}_h$ have the same edge structure, i.e., either they are both open-closed or both closed-open.

The abbreviations in Formulae (2), (3) and (4) are used in Table 6 to encode the channel-based, the broadcast and the one-to-many synchronizations. The intermediate CLTLoc formula ϕ_{sync_type} (where *sync_type* is *channel*, *broadcast* or *one-to-many*) is $\phi_{sync\_type} := \mathcal{G}(\upsilon_1 \wedge \upsilon_2)$, where υ_1 and υ_2 depend on the selected type of synchronization. Then, if multiple synchronizations are considered, ϕ_s is the conjunction of the corresponding ϕ_{sync_type}, one for each type of synchronization. Note that the syntax of MITL adopted in this work does not allow event symbols to appear in the formulae, being the language limited to constrain the values of the variables in *Int* and the atomic propositions in *AP* over the time. For this reason, symbols of Act_τ do not have a corresponding CLTLoc representation, yet they are used to instantiate the formulae encoding the synchronization among the TA.

- *Channel-based synchronization.* Formula υ_1 specifies that any sending event $\alpha!$ in a transition t of an automaton k must be matched by exactly one corresponding receiving event $\alpha?$ of a transition t' of another automaton h. This is specified by stating that there exists one of the automata with index h that syncs on $\alpha?$ and all the others (with index different than k and h) do not sync on $\alpha?$. Furthermore, the shape of the edges of the transitions of the automata that sync on action α must correspond.

Table 6. Formulae encoding different types of synchronizations.

Name	Property
Channel	$v_1 := \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha!}} \left(t[k]=t \rightarrow \bigvee_{h\in[1,K], h\neq k} \begin{pmatrix} \phi_{\textsf{sync-on}}(h,\alpha?) \wedge \neg\phi_{\textsf{sync-on-but}}(\{k,h\},\alpha?) \\ \wedge \\ \phi_{\textsf{same-edge}}(k,h) \end{pmatrix} \right)$ $v_2 := \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha?}} \left(t[k]=t \rightarrow \bigvee_{h\in[1,K], h\neq k} \left(\phi_{\textsf{sync-on}}(h,\alpha!) \wedge \neg\phi_{\textsf{sync-on-but}}(\{k,h\},\alpha!)\right) \right)$
Broadcast	$v_1 := \begin{array}{c} \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha\#}} \left(\mathrm{t}[k]=t \rightarrow (\neg\phi_{\textsf{sync-on-but}}(\{k\},\alpha\#))\right) \\ \wedge \\ \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha\#}} \left(\mathrm{t}[k]=t \rightarrow \left(\bigwedge_{\substack{h\in[1,K]\\ h\neq k}} \begin{pmatrix} \phi_{\textsf{sync-on}}(h,\alpha@) \wedge \phi_{\textsf{same-edge}}(k,h) \\ \vee \\ \left(\bigwedge_{t'\in T_{k'} \mid t'_e=\alpha@} (\mathcal{X}(\neg\phi_{t'_\gamma}) \vee \neg\phi_{t'_{\gamma var}} \vee l[h]\neq t'^{-}) \right) \end{pmatrix} \right) \right) \end{array}$ $v_2 := \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha@}} \left(t[k]=t \rightarrow \phi_{\textsf{sync-on-but}}(\{k\},\alpha\#)\right)$
One-to-many	$v_1 := \bigwedge_{\substack{k\in[1,K]\\ t\in T_k \mid t_e=\alpha\&}} \left(\mathrm{t}[k]=t \rightarrow (\neg\phi_{\textsf{sync-on-but}}(\{k\},\alpha\&) \wedge \phi_{\textsf{sync-on-but}}(\{k\},\alpha*))\right)$ $v_2 := \bigwedge_{\substack{k\in[1,K],\\ t\in T_k \mid t_e=\alpha*}} \left(t[k]=t \rightarrow \phi_{\textsf{sync-on-but}}(\{k\},\alpha\&)\right)$

Formula v_2 specifies that any receiving event $\alpha?$ must be matched by exactly one corresponding sending event $\alpha!$ in one of the other automata.

- *Broadcast synchronization.* Formula v_1 first specifies that if an automaton k broadcasts on a channel α, no other automaton broadcasts on that channel. Then, it specifies that if an automaton k broadcasts on a channel α, all the other automata h either sync and receive on that channel, and also match the shape of the transition, or they do not sync. If the automaton does not sync, either it is in a state that has no outgoing transition labeled with $\alpha@$, or the guards of the outgoing transitions labeled with $\alpha@$ are not satisfied (i.e., no transitions are enabled).
  Formula v_2 specifies that any receiving event $\alpha@$ must be matched by exactly one corresponding sending event $\alpha\#$ in one of the other automata.
- *One-to-many.* Formula v_1 specifies that if an event $\alpha\&$ is sent, no other automaton sends the same event, and at least one automaton receives the event $\alpha*$. Formula v_2 specifies that if an event $\alpha*$ is received by an automaton, some automaton has sent event $\alpha\&$.

As mentioned in Remark 6, the semantics—and corresponding encoding—of networks of TA allows for transitions with different types of edged to be taken at the same time. As depicted in Fig. 5, it also allows the same automaton to take transitions with different edges over time. However, one might desire to restrict this behavior (for example for synchronization reasons), and only allow transitions to be taken with a certain type of edge. These restrictions (if any), are captured by

Table 7. Formulae encoding different types of edges.

Name	Property
closed-open	$\mathcal{G} \bigwedge_{k[1,K]} \mathsf{edge}^{](}[k]$
open-closed	$\mathcal{G} \bigwedge_{k[1,K]} \neg\mathsf{edge}^{](}[k]$
unrestricted	$\top$

formula φ_{ef}. Table 7 shows some examples of restrictions and corresponding CLTLoc constraints. In particular, the "closed-open" (resp., open-closed) restriction states that, when a transition is taken, it must with some symbol $\alpha^{](}$ (resp., $\alpha^{)[}$). The "unrestricted" case obviously means that no constraint is introduced, hence it corresponds to true. Other possibilities could be envisaged, but these are the most relevant for our purposes.

The following theorem extends Theorem 4.3 to liveness conditions and synchronization mechanisms.

Theorem 4.4. *Let $\mathcal{N}$ be a network of TA, l be a (set of) liveness conditions, s be the (set of) synchronization mechanisms used in network $\mathcal{N}$, and ef the set of restrictions on edges. Let $\Phi_{\mathcal{N}}$ be the CLTLoc formula $\varphi_{\mathcal{N}} \wedge \varphi_l \wedge \varphi_s \wedge \varphi_{ef}$.*

For every trace η of $\mathcal{N}$ that satisfies the selected liveness conditions l, synchronization mechanisms s, and edge restrictions edge, any CLTLoc model (π, σ, ι) such that $(\pi, \sigma, \iota) \in \rho(\eta)$ is a model of $\Phi_{\mathcal{N}}$.

Conversely, for each CLTLoc model (π, σ, ι) of $\Phi_{\mathcal{N}}$, $\rho^{-1}((\pi, \sigma, \iota))$ is a trace of $\mathcal{N}$ that satisfies the selected liveness conditions l, synchronization mechanisms s, and edge restrictions ef.

The proof is omitted for reasons of brevity, as it is rather standard. Indeed, it is a straightforward extension of the proof of Theorem 4.3, and follows from the fact that each CLTLoc formula listed in Table 5 (resp., Table 6) encodes the corresponding semantics described in Table 1 (resp., Table 2); in addition, the formulae of Table 7 capture the corresponding restrictions on edges.

5 CHECKING THE SATISFACTION OF MITL FORMULAE OVER TA

Traces encode executions of TA by means of denumerable sequences of time and discrete transitions. However, the evolution of a network of TA is continuous, hence it is more naturally represented by means of *signals* (see Section 2.2). Timed words, instead of signals, are commonly adopted to represent the semantics of TA: although they are expressive enough in many cases, they cannot describe the values at the edge of signals—i.e., in correspondence of configuration changes. For instance, in a temporal logic such as MITL one can indeed state properties whose value is affected by signal edges; e.g., the set of signals of symbol p that change value only over intervals that are left-closed/right-open can be specified with the MITL formula $\mathcal{G}_{[0,+\infty)} \big((p \Rightarrow p\,\mathcal{U}_{(0,+\infty)} \top) \wedge (\neg p \Rightarrow \neg p\,\mathcal{U}_{(0,+\infty)} \top)\big)$. Therefore, model checking a TA against such an expressive language requires modeling edges as well.

Traces are tightly bound to signals: intuitively, given a trace η, the projection over the real line of the values of its integer variables and atomic propositions determines a signal M_η. To be able to consistently associate signals with traces of a TA, however, we need to impose the following restriction on traces.

Definition 5.1. Let $\mathcal{N}$ be a network of TA. A trace η of $\mathcal{N}$ is *edge-consistent* if, for any configuration change $(\mathrm{l}'_h, v'_{\mathrm{var},h}, v'_h) \xrightarrow{\Lambda_h} (\mathrm{l}_{h+1}, v_{\mathrm{var},h+1}, v_{h+1})$ there are two transitions $\mathrm{l}'_h[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} \mathrm{l}'_{h+1}[k]$

and $\mathtt{l}'_h[\bar{k}] \xrightarrow{\bar{\gamma},\bar{\xi},\bar{\alpha},\bar{\zeta},\bar{\mu}} \mathtt{l}'_{h+1}[\bar{k}]$, of two distinct TA $k, \bar{k}$, which both set the value of variable n (in a compatible manner), then the edge of the transitions is the same; that is, either they are $\alpha^{](}$ and $\bar{\alpha}^{](}$, or they are $\alpha^{)[}$ and $\bar{\alpha}^{)[}$.

In the rest of this section, only traces that are edge-consistent are considered.

Let η be an edge-consistent trace $(\mathtt{l}_0, v_{\mathrm{var},0}, v_0) \xrightarrow{\delta_0} (\mathtt{l}'_0, v'_{\mathrm{var},0}, v'_0) \xrightarrow{\Lambda_0} (\mathtt{l}_1, v_{\mathrm{var},1}, v_1) \xrightarrow{\delta_1} \ldots$; we indicate by $\Upsilon(e)$ the "time" of a symbol e (where e can be either δ or Λ), defined as follows:

- $\Upsilon(\delta_0) = 0$;
- $\Upsilon(\Lambda_h) = \Upsilon(\delta_h) + \delta_h$ for all $h \geq 0$;
- $\Upsilon(\delta_h) = \Upsilon(\Lambda_{h-1})$ for all $h > 0$.

Recall that, given a trace η, its associated word $w(\eta)$ is the sequence $\delta_0\Lambda_0\delta_1\Lambda_1\ldots$; also, given a set of assignments μ, $U(\mu)$ is the set of variables updated by μ. Let $(\mathtt{l}, v_{\mathrm{var}}, v)$ be a configuration; we denote as $c(\mathtt{l}, v_{\mathrm{var}}, v)$ the pair $(\cup_{1\leq k\leq K} L(\mathtt{l}_h[k]), v_{\mathrm{var},h}) \in \wp(AP) \times \mathbb{Z}^{Int}$ of the atomic propositions and variable assignments that hold in the configuration $(\mathtt{l}, v_{\mathrm{var}}, v)$.

Definition 5.2. Let η be an edge-consistent trace of a network $\mathcal{N}$ of TA. The *signal* M_η associated with η is the function $M_\eta : \mathbb{R}_{\geq 0} \to \wp(AP) \times \mathbb{Z}^{Int}$ such that:

(1) $M_\eta(0) = c(\mathtt{l}_0, v_{\mathrm{var},0}, v_0)$;
(2) for all δ_h in $w(\eta)$, for all $r \in \mathbb{R}_{\geq 0}$ such that $\Upsilon(\delta_h) < r < \Upsilon(\delta_h) + \delta_h$ then $M_\eta(r) = c(\mathtt{l}_h, v_{\mathrm{var},h}, v_h)$;
(3) for all Λ_h in $w(\eta)$, $M_\eta(\Upsilon(\Lambda_h)) = (A, v_{\mathrm{var}}) \in \wp(AP) \times \mathbb{Z}^{Int}$ where, for all $p \in AP$ and $n \in Int$:
  (a) $p \in A$ if, for some $\alpha \in Act_\tau$ and for some $1 \leq k \leq K$:
    - $p \in L(\mathtt{l}_h[k])$ and $\Lambda_h[k] \in \{\_, \alpha^{](}\}$ holds, or
    - $p \in L(\mathtt{l}_{h+1}[k])$ and $\Lambda_h[k] = \alpha^{)[}$ holds
  (b) $v_{\mathrm{var}}(n) = v_{\mathrm{var},h}(n)$ if one of the following conditions holds:
    - there is no transition $\mathtt{l}'_h[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} \mathtt{l}_{h+1}[k]$ compatible with the configuration change and such that $n \in U(\mu)$;
    - there is $1 \leq k \leq K$ and a transition $\mathtt{l}'_h[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} \mathtt{l}_{h+1}[k]$—compatible with the configuration change—such that $\Lambda_h[k] = \alpha^{](}$ and $n \in U(\mu)$.
  (c) $v_{\mathrm{var}}(n) = v_{\mathrm{var},h+1}(n)$ if there is $1 \leq k \leq K$ and a transition $\mathtt{l}'_h[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} \mathtt{l}_{h+1}[k]$—compatible with the configuration change—such that $\Lambda_h[k] = \alpha^{)[}$ and $n \in U(\mu)$ hold.

Condition (2) defines the correspondence between the time transitions in a trace η and the values of the signal within the left-open/right-open intervals of M_η. In particular, any trace η defines an infinite set of intervals I_h of the form $(\Upsilon(\delta_h), \Upsilon(\delta_h) + \delta_h)$, for all $h \geq 0$. The value of signal M_η in every interval I_h is determined by the propositions and variable assignments $c(\mathtt{l}_h, v_{\mathrm{var},h}, v_h)$ that hold in I_h.

Condition (3) handles the case of a discrete transition in h and defines the value of M_η at time $\Upsilon(e_h)$ when a configuration change occurs (i.e., when $e_h = \Lambda_h$ holds). Conditions (3)a, (3)b and (3)c define, respectively, the atomic propositions and the value of the integer variables based on the transition $\mathtt{l}'_h[k] \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} \mathtt{l}_{h+1}[k]$ performed at time $\Upsilon(\Lambda_h)$ by automaton $\mathcal{A}_k$, for every $1 \leq k \leq K$. Condition (3)a specifies that $M_\eta(\Upsilon(\Lambda_h))$ includes the atomic propositions in $L(\mathtt{l}_h[k])$ if the discrete transition performed by $\mathcal{A}_k$ is closed-open (i.e., $\Lambda_h[k] = \alpha^{](}$ holds), or if no transition is taken; otherwise, if the discrete transition is open-closed (i.e., $\Lambda_h[k] = \alpha^{)[}$ holds), $M_\eta(\Upsilon(\Lambda_h))$ includes the atomic propositions in $L(\mathtt{l}_{h+1}[k])$. Conditions (3)b and (3)c define the value $v_{\mathrm{var}}(n)$ of variable n at time $\Upsilon(\Lambda_h)$. The value of $v_{\mathrm{var}}(n)$ is the same as $v_{\mathrm{var},h}(n)$ if there is an automaton $\mathcal{A}_k$ that

Table 8. Definition of different types of signals based on the symbols occurring in the corresponding traces.

Signal	**Discrete transitions**
right-closed	$\alpha^{](}$
left-closed	$\alpha^{)[}$
unrestricted	$\alpha^{](}$ or $\alpha^{)[}$

performs a closed-open discrete transition that modifies n, or if none of the automata updates n (condition (3)b). Conversely (condition (3)c), the value of n becomes $v_{\mathrm{var},h+1}(n)$ at time $\Upsilon(\Lambda_h)$ if there is an automaton $\mathcal{A}_k$ that performs a open-closed discrete transition that modifies n.

Consider a trace η and its associated signal M_η. M_η is *left-closed* when, for all $r \in \mathbb{R}_{\geq 0}$, if $M_\eta(r) = (A, v_{\mathrm{var}})$ for some A, v_{var}, then there is $\varepsilon \in \mathbb{R}_{>0}$ such that, for all $r < r' < r + \varepsilon$ it also holds $M_\eta(r') = (A, v_{\mathrm{var}})$. Dually, M_η is *right-closed* when, for all $r \in \mathbb{R}_{>0}$, if $M_\eta(r) = (A, v_{\mathrm{var}})$ for some A, v_{var}, then there is $\varepsilon \in \mathbb{R}_{>0}$ such that, for all $r - \varepsilon < r' < r$ it also holds $M_\eta(r') = (A, v_{\mathrm{var}})$. M_η is *unrestricted* if there are no constraints on the value of the signal in the neighborhood of each time instant.

The shape of a signal M_η is determined by the transitions taken by the automata of the network. Consider, for instance, a variable n with value 2, and a transition $t = q \xrightarrow{\gamma,\xi,\alpha,\zeta,\mu} q'$ that, when it is taken, assigns value 1 to n. There are different possibilities concerning the value of variable n in the instant when t is taken: if it must be 2 (i.e., it is not yet assigned by the transition), then the corresponding signal cannot be left-closed; if it must be 1 (i.e., it is already assigned by the transition), then the signal cannot be right-closed. There is an obvious relation (captured by Table 8) between a signal M_η being right-closed, left-closed, or unrestricted, and the edges of the transitions taken in η. Indeed, when all edges are open-closed (i.e., $\alpha^{)[}$), the signal is left-closed; when they are all closed-open (i.e., $\alpha^{](}$) the signal is right-closed; when they can be both, the signal is unrestricted. Leaving signals unrestricted—which means that, when transitions are fired, the choice of whether variables are already assigned their new values or still retain their old ones is non-deterministic—is a common approach in literature that has been used in some seminal works on TA [5].

Then, by imposing constraints on the types of edges that the transitions of a network $\mathcal{N}$ of TA can have, one can restrict the set of corresponding signals to contain only left-closed or only right-closed ones.

Definition 5.3. Let $\mathcal{N}$ be a network of TA, l be a set of liveness conditions selected from Table 1, s be the synchronization primitives (among those of Table 2) used in $\mathcal{N}$, and ef a restriction on the types of edges for the transitions taken by $\mathcal{N}$ (such as those of Table 8).

$\mathcal{T}(\mathcal{N}, l, s, ef)$ is the set of edge-consistent traces that satisfy the liveness conditions l, the semantics of synchronizations s, and the restriction on edges ef.

In addition, $\mathcal{S}(\mathcal{N}, l, s, ef)$ is the corresponding set of signals.

Example 5.4. Figure 6 shows the same trace of Fig. 5 and the values of l and n that contribute to the definition of the signal deriving from the trace. At the bottom, three different types of signals—as specified on the right-hand side of the figure—are drawn according to the selected restrictions on the edges. The last signal is based on the assignment of variable edge in Fig. 5, whereas the first two are derived by restricting the kind of discrete transitions of the traces. Similarly to Fig. 5, the value of edge is explicitly written by means of](or)[. In correspondence to the events e_1, e_2 and e_3 its value defines the edge of the current interval, while it is left unspecified in the other positions to avoid cluttering the figure.

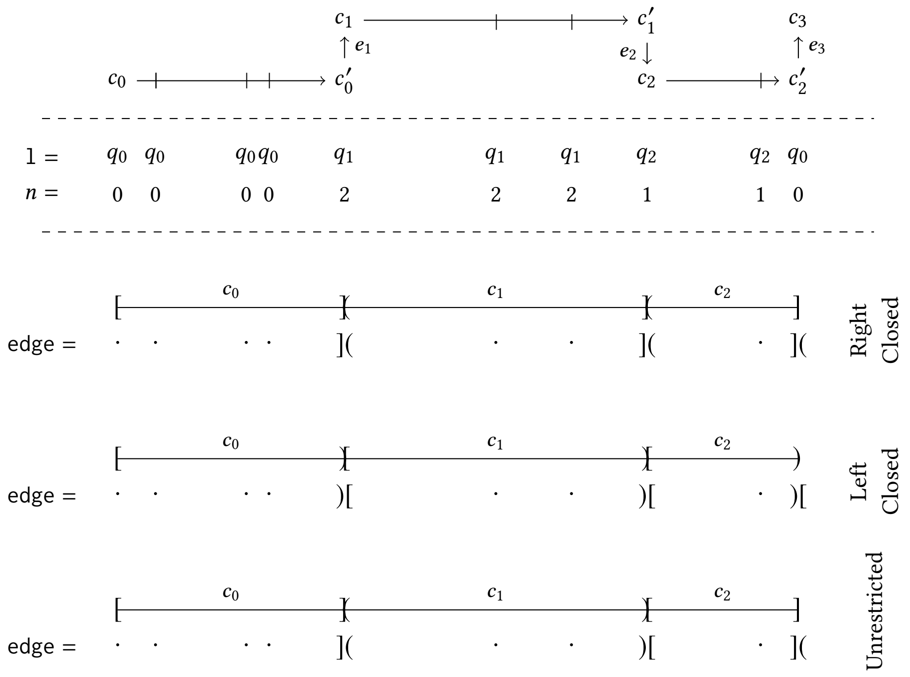


Fig. 6. Illustration of the right-closed, left-open and unrestricted semantics on the trace presented in Fig. 5 generated by the TA of Fig. 2(b).

5.1 Verification problem of networks of TA with respect to MITL formulae

The properties of networks of TA are encoded by formulae that predicate over the values of the variables of set Int and over the atomic propositions which are labeling locations. This section defines when a network $\mathcal{N}$ of TA satisfies a MITL property ψ and states the verification problem of networks of TA against a MITL formula. Both definitions are based on the idea of selecting a (possibly proper) subset of traces of $\mathcal{N}$, with respect to some selection criterion T.

Given a network $\mathcal{N}$ of TA, a selection criterion T for the traces of $\mathcal{N}$ identifies a subset of traces of $\mathcal{N}$. An example of selection criterion T could be "the set of traces that correspond to right-closed signals". A trivial selection criterion simply identifies the set of all traces of $\mathcal{N}$. With a slight abuse of notation, in the following, T indicates both the selection criterion and the set of traces that it identifies.

Definition 5.5 (Satisfiability of MITL formulae over networks of TA). Let $\mathcal{N}$ be a network of TA, T a selection criterion, and ψ a MITL formula. $\mathcal{N}$ satisfies ψ restricted to T (written $\mathcal{N} \models_T \psi$) if every signal $M_\eta \in T$ is such that $M_\eta, 0 \models \psi$ holds.

Definition 5.6 (Verification problem). Let $\mathcal{N}$ be a network of TA, T be a selection criterion, and ψ be a MITL formula. The verification problem for the network of TA $\mathcal{N}$ restricted to T against a MITL formula ψ consists in determining whether $\mathcal{N} \models_T \psi$ holds.

In the rest of this paper, the adopted selection criteria restrict the traces of interest to those that satisfy some liveness conditions l, the semantics of the synchronization primitives s appearing in $\mathcal{N}$, and some restriction ef on the edges of the transitions taken. Such a selection criterion is denoted

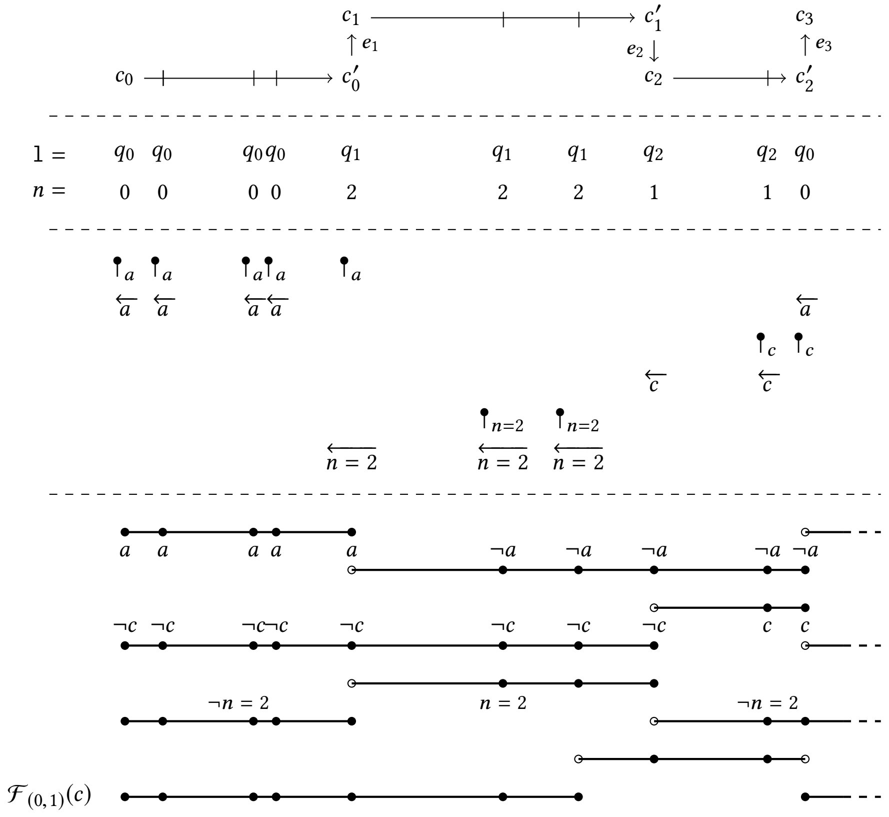


Fig. 7. Relationship between a trace and the MITL signal derived by Formula φ_{sig}.

as $T = \langle l, s, ef \rangle$. In this case, the set of selected traces is $\mathcal{T}(\mathcal{N}, l, s, ef)$, and the corresponding signals are $\mathcal{S}(\mathcal{N}, l, s, ef)$.

In the following, the verification problem of Def. 5.6 is reduced to the problem of checking the satisfiability of CLTLoc formula $\Phi_{sig} \wedge \Phi_{\neg\psi}$, where Φ_{sig} and $\Phi_{\neg\psi}$ are computed as specified in Sections 5.3 and 5.2, respectively. Section 5.4 shows the correctness of the proposed procedure.

5.2 CLTLoc encoding of MITL signals

Bersani et al. [9] showed how to build a CLTLoc formula Φ_ψ from a MITL formula ψ such that the set M_ψ of signals that are models of ψ (i.e., $M_\psi = \{M | M, 0 \models \psi\}$) is represented by the set of models of Φ_ψ—hence, the satisfiability of ψ is reduced to the satisfiability of Φ_ψ. Mapping a continuous-time signal M to a denumerable sequence of elements is done by partitioning $\mathbb{R}_{\geq 0}$ into infinitely many bounded intervals, each one representing a portion of M in which the values of propositions and integer variables do not change (except possibly in the endpoints). In particular, let I be an interval of the form (a, b), with $a < b$, and $I_0, I_1, \ldots$ be a denumerable set of adjacent intervals (i.e., $a_{i+1} = b_i$ holds for all $i > 0$) covering $\mathbb{R}_{\geq 0}$—i.e., such that $\bigcup_{i \geq 0}(I_i \cup \{a_i\}) = \mathbb{R}_{\geq 0}$ holds, with $a_0 = 0$. Every position i in a CLTLoc model of Φ_ψ represents the "configuration of ψ"—i.e., the value of all its subformulae—in interval I_i and at instant a_i, according to the semantics of MITL.

Remark 7. *As already remarked in Sec. 3, every sequence of consecutive time transitions in a trace can be replaced by an equivalent sequence of alternating time and discrete transitions, such that the total amount of elapsed time is the same as the original time transition and in all introduced discrete transitions every automaton does not perform any configuration change. Every position of the models of $\Phi_{sig} \wedge \Phi_{\neg\psi}$ represents a time instant where either the configuration of $\mathcal{N}$, or the configuration of $\neg\psi$, changes, or both possibly change at the same time. Therefore, in case $\neg\psi$ changes configuration at position h—i.e., one of its subformulae changes value—but $\mathcal{N}$ does not, then h in the trace of $\mathcal{N}$ corresponds to a discrete transition Λ_h such that $\Lambda_i[k] = \_$, for all $1 \le k \le K$. Figure 7, which will be discussed more in depth in Example 5.8, exemplifies this situation by showing a formula that changes value while the automaton does not take any transition.*

5.3 CLTLoc encoding of network signals

Let $\mathcal{N}$ be a network of TA, l be a set of liveness conditions selected from Table 1, s be the semantics of the synchronization primitives appearing in $\mathcal{N}$ (selected from Table 2), and ef be a restriction on the edges of the transitions taken by $\mathcal{N}$; this section defines the CLTLoc formula Φ_{sig} representing the set of signals in $\mathcal{S}(\mathcal{N}, l, s, ef)$. By Def. 5.2, every trace η is associated with a signal M_η that can be decomposed into the initial value $M_\eta(0)$, an infinite set of intervals $(\Upsilon(\delta_0), \Upsilon(\delta_1)), (\Upsilon(\delta_1), \Upsilon(\delta_2)), \ldots$, defined by the time transitions δ_h, and a set of time instants $\Upsilon(\Lambda_h)$ corresponding to discrete transitions with symbol Λ_h, where $h \ge 0$.

According to [9], suitable CLTLoc atoms can be used to represent the signal defined by the atomic propositions labeling the locations of automata and the arithmetical formulae occurring in a MITL formula. In the following, AF indicates the universe of propositions of the form $n \sim d$, where n is an integer variable and d is a constant. For every $\beta \in AP \cup AF$, the value of β in the intervals $(\Upsilon(\delta_h), \Upsilon(\delta_{h+1}))$ is represented by proposition $\overleftarrow{\beta}$, called *rest* of β; similarly, the value of β in time instants $\Upsilon(\Lambda_h)$ is represented by a proposition $\overset{\bullet}{\shortmid}_{\beta}$, called *first* of β.

Formula Φ_{sig} is built by combining formula $\Phi_{\mathcal{N}}$ defined in Theorem 4.4, representing the traces η of $\mathcal{N}$, and a formula that constrains the propositions $\overset{\bullet}{\shortmid}_{\beta}$ and $\overleftarrow{\beta}$, so that the signal M_η is correctly defined and all the conditions in Def. 5.2 are satisfied. Even though a signal M_η specifies a valuation v_{var} in every time instant, and it defines the exact assignment for every variable $n \in Int$, formula Φ_{sig} only represents the signal of the formulae $n \sim d$ that appear in the MITL formula ψ, because the truth of ψ is determined only by the value of its subformulae. The value of $\overset{\bullet}{\shortmid}_{n\sim d}$ and $\overleftarrow{n \sim d}$, however, is defined in every time position i of the model of Φ_{sig} by the value $\iota(i, n)$.

Formula Φ_{sig} is defined in (5). It is composed by two parts, formula $\Phi_{\mathcal{N}}$ and formula φ_{sig}, which maps traces to signals as defined in (6).

$$\Phi_{sig} := \overbrace{\varphi_{\mathcal{N}} \wedge \varphi_l \wedge \varphi_s \wedge \varphi_{ef}}^{\Phi_{\mathcal{N}}} \wedge \varphi_{sig} \tag{5}$$

Recall that, as defined in Formula (1), $\varphi_{\mathcal{N}}$ encodes the network of TA in CLTLoc; in addition, φ_l, φ_s and φ_{ef} impose the liveness conditions, synchronization mechanisms and restrictions on edges by means of the formulae in Tables 5, 6 and 7. Finally, formula φ_{sig} is defined as follows, through the formulae of Table 9.

$$\varphi_{sig} := \bigwedge_{i \in [1,6]} \chi_i \tag{6}$$

Formulae χ_1–χ_6 create a mapping between the values of the atomic propositions and of the variables and the corresponding signals. More precisely, formulae χ_3 and χ_4 (resp., χ_1 and χ_2) bind the values

Table 9. Formulae encoding the relation between the network of TA and the signal at the initial time instant and within the intervals.

$\chi_1 \coloneqq \bigwedge_{\substack{k\in[1,K],\\ p\in AP}} \left(\text{⫯}_p \leftrightarrow \bigvee_{p\in L(q_{0,k})} \mathsf{l}[k] = q_{0,k} \right)$	$\chi_2 \coloneqq \bigwedge_{(n\sim d)\in AF} \left(\text{⫯}_{n\sim d} \leftrightarrow n \sim d \right)$
$\chi_3 \coloneqq \mathcal{G} \bigwedge_{p\in AP} \left(\overleftarrow{p} \leftrightarrow \bigvee_{k\in[1,K], q\in Q_k, p\in L(q)} \mathsf{l}[k] = q \right)$	$\chi_4 \coloneqq \mathcal{G} \bigwedge_{(n\sim d)\in AF} \left(\overleftarrow{n \sim d} \leftrightarrow n \sim d \right)$
$\chi_5 \coloneqq \mathcal{G} \bigwedge_{p\in AP} \left(\mathcal{X}(\text{⫯}_p) \leftrightarrow \left(\begin{array}{c} \bigvee_{k\in[1,K], q\in Q_k, p\in L(q)} \mathsf{l}[k] = q \wedge \left(\mathsf{t}[k] = \natural \vee (\mathsf{t}[k] \neq \natural \wedge \mathsf{edge}^{](}[k]) \right) \\ \vee \\ \bigvee_{k\in[1,K], q\in Q_k, p\in L(q)} \mathcal{X}(\mathsf{l}[k] = q) \wedge \mathsf{t}[k] \neq \natural \wedge \neg\mathsf{edge}^{](}[k] \end{array} \right) \right)$	
$\chi_6 \coloneqq \mathcal{G} \bigwedge_{(n\sim d)\in AF} \left(\mathcal{X}(\text{⫯}_{n\sim d}) \leftrightarrow \left(\begin{array}{c} n \sim d \wedge \left(\begin{array}{c} \neg \bigvee_{k\in[1,K], t\in T_k, n\in U(t)} \mathsf{t}[k] = t \\ \vee \\ \bigvee_{k\in[1,K], t\in T_k, n\in U(t)} \mathsf{t}[k] = t \wedge \mathsf{edge}^{](}[k] \end{array} \right) \\ \vee \\ \mathcal{X}(n \sim d) \wedge \bigvee_{k\in[1,K], t\in T_k, n\in U(t)} \mathsf{t}[k] = t \wedge \neg\mathsf{edge}^{](}[k] \end{array} \right) \right)$	

of the atomic propositions and of the variables to the corresponding signal within each time interval $(\Upsilon(\delta_h), \Upsilon(\delta_{h+1}))$ (resp., in the origin). Formulae χ_5 and χ_6, instead, bind the values of the atomic propositions and of the variables to the corresponding signals at the boundaries of the intervals—i.e., at time instants $\Upsilon(\Lambda_h)$.

Lemma 5.7. *Let $\mathcal{N}$ be a network of TA, l be a set of liveness conditions selected from Table 1, s be the semantics of the synchronization primitives appearing in $\mathcal{N}$ (selected from Table 2), ef be a restriction on the edges of the transitions taken by $\mathcal{N}$ (from Table 8), and Φ_{sig} be the corresponding CLTLoc formula (5).*

For every edge-consistent trace η of $\mathcal{N}$ that also belongs to $\mathcal{T}(\mathcal{N}, l, s, ef)$, and whose associated signal is M_η, there exists a model (π, σ, ι) of Φ_{sig} such that:

(1) for every time instant $r \in \mathbb{R}_{\geq 0}$ such that $\Upsilon(\delta_h) < r < \Upsilon(\delta_{h+1})$, for some $h \in \mathbb{N}$, if $M_\eta(r) = (P, v_{\text{var}})$, the following conditions hold:

$$p \in P \quad \textit{iff} \quad (\pi, \sigma, \iota), h \models \overleftarrow{p} \tag{7}$$

$$v_{var}(n) \sim d \quad \textit{iff} \quad (\pi, \sigma, \iota), h \models \overleftarrow{n \sim d} \tag{8}$$

(2) for every time instant $r \in \mathbb{R}_{\geq 0}$ such that $r = \Upsilon(\Lambda_h)$, for some $h \in \mathbb{N}$, if $M_\eta(r) = (P, v_{\text{var}})$, the following conditions hold:

$$p \in P \quad \textit{iff} \quad (\pi, \sigma, \iota), h + 1 \models \text{⫯}_p \tag{9}$$

$$v_{\text{var}}(n) \sim d \quad \textit{iff} \quad (\pi, \sigma, \iota), h + 1 \models \text{⫯}_{n\sim d} \tag{10}$$

Conversely, for every model (π, σ, ι) of Φ_{sig}, there exists an edge-consistent trace η of $\mathcal{N}$ that belongs to set $\mathcal{T}(\mathcal{N}, l, s, ef)$, with associated signal M_η, for which conditions (1) and (2) hold.

Sketch of Proof. Let η be an edge-consistent trace of $\mathcal{N}$ that also belongs to set $\mathcal{T}(\mathcal{N}, l, s, ef)$. By Theorem 4.4, every (π, σ, ι) such that $(\pi, \sigma, \iota) \in \rho(\eta)$ is a model of $\Phi_{\mathcal{N}}$. Formula $\Phi_{\mathcal{N}}$ does not constrain propositions ⫯_β and $\overleftarrow{\beta}$, with $\beta \in AP \cup AF$. Hence, it has to be proven that, if (π, σ, ι), and M_η are also such that conditions (7)-(10) hold, then (π, σ, ι) is a model of Φ_{sig}. Since (π, σ, ι) is a model for $\Phi_{\mathcal{N}}$ (Thm. 4.3), it is enough to show that (π, σ, ι) is a model also for φ_{sig}. By definition, η meets the conditions of Def. 5.2.

It is straightforward to show that subformulae χ_1 and χ_2 of φ_{sig} hold because of condition (1) of Def. 5.2, since they state that in the origin of the signal predicates ⫯_β and $\overleftarrow{\beta}$, for $\beta \in AP \cup AF$, correspond to the initial configuration of $\mathcal{N}$. Similarly, subformulae χ_3 and χ_4 hold because of condition (2), since they capture the fact that, in each interval $(\Upsilon(\delta_h), \Upsilon(\delta_{h+1}))$, the predicates that hold are those of position h of (π, σ, ι), which derives, by mapping ρ, from configuration $(\mathrm{l}_h, v_{\mathrm{var},h}, v_h)$.

Consider now formula χ_5. The first disjunct of the right-hand side states that the label p holds at the beginning of an interval I_{h+1}—i.e., at time instant $\Upsilon(\Lambda_h)$, for $h \geq 0$—if it held in the previous interval I_h f or an automaton $\mathcal{A}_k$, and either $\mathcal{A}_k$ does not take any transition (i.e., $\Lambda_h[k] = \_$) or, if takes one transition, it does so with an](edge (i.e., $\Lambda_h[k] = \alpha^{](}\}$); this corresponds to the first bullet of condition (3)a of Def. 5.2. The second disjunct, instead, states that p holds at the beginning of an interval I_{h+1} if there is an automaton $\mathcal{A}_k$ that takes a transition, and it does so with an)[edge (i.e., $\Lambda_h[k] = \alpha^{)[}\}$), which corresponds to the second bullet of condition (3)a of Def. 5.2. Similarly, the first disjunct of the right-hand side of formula χ_6 captures condition (3)b of Def. 5.2 (each disjunct in the subformula corresponds to one of the bullets of condition (3)b), while the second disjunct captures condition (3)c.

The second part of the statement is proven by showing that, given a model (π, σ, ι) of Φ_{sig}, the corresponding trace $\eta = \rho^{-1}((\pi, \sigma, \iota))$ is such that conditions (7)-(10) hold for signal M_η. This can be done using similar arguments as those presented in the first part of the proof, and is omitted for brevity. □

5.4 Model-checking of networks of TA with respect to MITL formulae

Lemma 5.7 establishes a correspondence between signals derived from traces of network $\mathcal{N}$ and models of formula Φ_{sig}. The models of formula Φ_{sig} include predicates of the type ⫯_β and $\overleftarrow{\beta}$, which act as a “bridge” with the encoding of MITL formulae that predicate over β. The next example shows how this allows us to match MITL constraints with signals derived from networks of TA.

Example 5.8. Figure 7 shows the relation between the trace of Example 4.2 depicted in Fig. 5 and the MITL signals referring to the atomic propositions a, c and the subformulae $n = 2$ and $\mathcal{F}_{(0,1)}(c)$. It shows the assignments to l and n, and the atoms representing the signals of a, c and $n = 2$ in correspondence to the positions where their value is true. For instance, at position 4, ⫯_a and $\overleftarrow{n = 2}$ hold, whereas ⫯_c, $\overleftarrow{c}$, $\overleftarrow{a}$ and $\text{⫯}_{n=2}$ are false. The bottom part of the figure shows the signals that are built according to the value of CLTLoc atoms ⫯_β and $\overleftarrow{\beta}$. The signal of each proposition is drawn on two levels: the top one represents the value true and the bottom one represents the value false. In every position, the value of the proposition is specified by a filled circle that defines the value in the exact time instant corresponding to the position and every line between adjacent positions represents the value of the proposition in the corresponding interval. An empty circle at the beginning or at the end of an interval indicates that the value of the formula in the interval does not extend also to the infimum or to the supremum of the interval, respectively. At position 6 formula $\mathcal{F}_{(0,1)}(c)$ changes value, because one time unit later (which corresponds to position 7 in this example) formula c becomes true; the automaton, instead, at position 6 has the same configuration—if clock assignments are not considered—as at position 5.

The next proposition shows how, given a network $\mathcal{N}$ of TA, a selection criterion T, and a MITL property ψ, the problem of checking whether $\mathcal{N} \models_T \psi$ holds can be reduced to that of determining the satisfiability of CLTLoc formula $\Phi_{sig} \wedge \Phi_{\neg\psi}$.

Proposition 1. *Let $\mathcal{N}$ be a network of TA, ψ be a MITL formula, and $T = \langle l, s, ef \rangle$—where l is a set of liveness conditions selected from Table 1, s is the semantics of the synchronizations primitives appearing in $\mathcal{N}$ (selected from Table 2), and ef is a restriction on the edges of the transitions taken by $\mathcal{N}$ (from Table 8). Also, let Φ_{sig} and Φ_{ψ} be the CLTLoc formulae defined in Section 5.3 and in Section 5.2, respectively. Then, $\mathcal{N} \models_T \psi$ holds if, and only if, $\Phi_{sig} \wedge \Phi_{\neg\psi}$ does not have any models.*

Sketch of Proof. By Lemma 5.7 and by the results of [9], $\Phi_{sig} \wedge \Phi_{\neg\psi}$ admits a model if, and only if, there is (π, σ, ι) that corresponds to a signal M_η that satisfies MITL formula $\neg\psi$, and such that trace η belongs to $\mathcal{T}(\mathcal{N}, l, s, ef)$. That is, $\Phi_{sig} \wedge \Phi_{\neg\psi}$ does not have any models if, and only if, $\mathcal{S}(\mathcal{N}, l, s, ef) \cap M_{\neg\psi} = \emptyset$ holds. This, in turn, is equivalent to saying that $\mathcal{S}(\mathcal{N}, l, s, ef) \subseteq M_{\psi}$ holds, which corresponds to Def. 5.5. □

Since there are automated tools for checking the satisfiability of CLTLoc formulae [6, 9], Proposition 1 establishes an effective technique to solve the verification problem of networks of TA with respect to MITL formulae: given a network $\mathcal{N}$ of TA, a selection criterion $T = \langle l, s, ef \rangle$, and a MITL formula ψ, it is enough to build CLTLoc formulae Φ_{sig} and $\Phi_{\neg\psi}$, then check the satisfiability of formula $\Phi_{sig} \wedge \Phi_{\neg\psi}$.

6 EVALUATION

The procedure proposed in Section 5.4 has been implemented in TACK (Timed Automata ChecKer)[5]. TACK is a Java 8 application that takes as input a model expressed using the Uppaal input format and a property expressed in MITL. The model and the property are converted in a CLTLoc formula as specified in Sect. 5. The satisfiability of the CLTLoc formula is verified using the Zot formal verification tool [6].

To evaluate TACK, a full direct comparison with existing tools, i.e., Uppaal, Mitl$_{0,\infty}$BMC [25], and MightyL [15], was not performed as such comparison would not be meaningful, for several reasons.

(i) Neither Uppaal, nor Mitl$_{0,\infty}$BMC fully support MITL. Uppaal supports a restricted subset of the TCTL logic, which allows the specification only of properties in the form: $\forall\,\mathcal{G}(e)$ ("for all executions e globally holds"); $\forall\,\mathcal{F}(e)$ ("for all executions e eventually holds"); $\exists\,\mathcal{G}(e)$ ("there exists an executions in which e globally holds"); $\exists\,\mathcal{F}(e)$ ("there exists an executions in which e eventually holds") and, finally, the so called "leads-to" formula which is encoded as $\forall\,\mathcal{G}(e \Rightarrow \forall\,\mathcal{F}(e'))$ ("in every execution it is always true that the occurrence of e always makes e' hold"), where e and e' are state formulae (i.e., expressions over state variables or automata locations). Mitl$_{0,\infty}$BMC, instead, considers the fragment MITL$_{0,\infty}$, but it does not provide any information on how the encoding can be extended to cover MITL. In fact, Mitl$_{0,\infty}$BMC adopts a super-dense semantics for time, requiring a suitable change of MITL semantics. Indeed, some useful properties [22][28] that hold for the standard MITL semantics, e.g., proving that the fragment MITL$_{0,\infty}$ has the same expressive

[5]The tool is available at http://github.com/claudiomenghi/TACK.

power as MITL, are not valid anymore over super-dense time. Therefore, extending the logical language used in $\text{Mitl}_{0,\infty}$BMC to MITL appears to be far from straightforward.

(ii) As mentioned above, both Uppaal and $\text{Mitl}_{0,\infty}$BMC [25] adopt the super dense semantics of time. This allows a TA to fire consecutive transitions without requiring time to progress. Uppaal introduces the syntactic notion of "committed locations" to prevent time from progressing—i.e., when an automaton is in a committed location, only action transitions can be fired and time cannot advance. The underlying notion of time adopted in this work is based on the CLTLoc semantics, which relies on the strict progress of time between adjacent positions and does not enable such modeling facility.

(iii) $\text{Mitl}_{0,\infty}$BMC is mainly a proof-of-concept tool that has not been further supported since 2013 and does not support a direct implementation of synchronization events ($\alpha!$ and $\alpha?$). The lack of suitable documentation (such as a user manual) does not allow a clear understanding of the potential offered by the tool.

(iv) MightyL [14, 15] allows users, at least in principle, to perform model checking of MITL formulae on TA using both the pointwise and the continuous (signal-based) semantics. However, the MightyL approach assumes that the model of the system to be checked is specified as a bipartite Signal Automaton (SA)—or, at the very least, as a TA that has the properties guaranteed by the transformation of SA into TA defined by Proposition 10 of [15]—, which then needs to be converted (together with the SA computed from the MITL formula) into a TA that can be checked using the LTSmin [24] model checker. However, the transformation from SA to TA defined by Proposition 10 of [15] is not supported by a publicly available tool. Moreover, TACK supports the verification of interacting networks of TA, whereas MightyL assumes that the model is captured through a single automaton, and the inclusion of synchronization primitives in models to be input to MightyL would further complicate the matter. Finally, only one example of model checking of MITL properties on TA through MightyL is presented in the literature [15], and for the reasons above we cannot adapt our own benchmarks to MightyL.

To summarize, since the capabilities of Uppaal and $\text{Mitl}_{0,\infty}$BMC are significantly different from those provided by TACK (especially in terms of the logic used to express the property), and performing model checking experiments with MightyL poses crucial obstacles, an exhaustive, direct comparison of TACK with these tools is not significant. Nevertheless, in addition to carrying out an extensive experimental evaluation of TACK, we also performed a limited set of experiments with $\text{Mitl}_{0,\infty}$BMC on one of our benchmarks, and we provided a brief, qualitative comparison of TACK and MightyL.

Concerning the experimental evaluation of TACK, we focused on the following features: (i) the efficiency of TACK in verifying MITL properties of TA; (ii) how TACK enables the introduction of the synchronization constructs and semantic constraints presented in Section 3. The ease of performing verification has been estimated through a bounded model checking technique that relies on two different solvers available in the Zot formal verification tool [6]. Both solvers check the satisfiability of CLTLoc formulae, but they are based on different techniques. They rely on SMT (Satisfiability Modulo Theories) solvers (Microsoft Z3 [18] in our case), as the satisfiability problem of CLTLoc has been tackled so far by reducing it to an SMT instance. The first solver, ae2Zot [11], reduces the satisfiability problem of CLTLoc formulae to that of a fragment of the first-order logic over real difference arithmetic; the second, ae2SBVZot [6], instead uses a Bit-Vector encoding. The ability of TACK to consider different features of TA is evaluated by selecting benchmarks that exploit different constructs such as, for example, different synchronization primitives.

Three different benchmarks are used in this first set of experiments focusing solely on TACK: the Fischer mutual exclusion protocol [3], the CSMA/CD protocol [1] and the Token Ring protocol [23]. All selected benchmarks have also been classically implemented in the Uppaal model checker [2].

TACK ran using the ae2Zot and ae2SBVZot solvers, version 4.7.1 of Z3, on a machine equipped with an Intel(R) Core(TM) i7-4770 CPU (3.40GHz) with 8 cores, 16GB of RAM and Debian Linux (version 8.8). To test the scalability of the approach, various configuration of the protocols were tested, each one determined by a different number n of involved agents. In particular, variable n indicates: for the Fischer benchmark, the number of participants; for the CSMA/CD protocol, the number of competing stations; for the Token Ring protocol, the number of processes. In the tests, n spans from 2 to 10. Since the CLTLoc solvers we used relied on a bounded model-checking approach, we considered a bound k spanning from 10 to 30, with increments of 5. For each combination of values of n and k we considered a timeout of 2 hours.

Fischer benchmark. This benchmark describes a mutual exclusion algorithm in which n participants try to enter a critical section. Before trying to enter the critical section, a participant first checks if another one is in the critical section. If this is not the case, it writes its (unique) identifier in a shared variable. After waiting a certain amount of time, it checks again the shared variable. If its identifier is still in the shared variable, it proceeds to the critical section. Otherwise, it goes back to start since another process had simultaneously checked whether the critical section was empty and set the shared variable. The synchronization among the participants is obtained through shared clocks and no synchronization on the transitions is present.

The following six properties (a subset of them was also considered in [25]) were verified.

$$
\begin{array}{lcl}
\textbf{live-one} & := & \mathcal{G}_{[0,\infty)}\left(p_1.req \rightarrow \mathcal{F}_{[0,\infty)}\, p_1.wait\right)\\
\textbf{live-two} & := & \mathcal{G}_{[0,\infty)}\left(p_1.req \rightarrow \mathcal{F}_{[0,3]}\, p_1.wait\right)\\
\textbf{live-three} & := & \mathcal{G}_{[0,\infty)}\left(p_1.req \rightarrow \mathcal{F}_{(0,3)}\, p1.cs\right)\\
\textbf{live-four} & := & \mathcal{G}_{[0,\infty)}\left(p_1.req \rightarrow \mathcal{F}_{(0,3)}\, p_1.wait\right)\\
\textbf{live-five} & := & \mathcal{G}_{[0,\infty)}\left(p_1.req \rightarrow \mathcal{F}_{[0,3]}\, p1.cs\right)\\
\textbf{live-six} & := & \mathcal{G}_{[0,\infty)}\left(\neg\left(\bigvee_{i=1:n-1}\left(p_i.cs \wedge \left(\bigvee_{j=i+1:n}\right) p_j.cs\right)\right)\right)
\end{array}
$$

Properties **live-one** and **live-six** are not metric, as $\mathcal{F}_{[0,\infty)}$ and $\mathcal{G}_{[0,\infty)}$ are equivalent to the LTL "eventually" and "globally" modalities. Properties **live-two** (resp., **live-three**) and **live-four** (resp., **live-five**) differ with respect to the interval specified in the $\mathcal{F}$ operator.

The results are presented in Table 10. The rows show the time (in seconds) required by the model-checking procedure with different bounds k. Symbol "−" indicates a timeout. The columns contain the results obtained by considering an increasing number n of participants. For properties **live-one**, **live-two**, **live-four** and **live-six** TACK always returned the correct result. For properties **live-three** and **live-five**, when k was equal to 10, the value of n is too large, compared to the bound k, to allow TACK to find a counterexample (i.e., to allow the underlying satisfiability procedure for CLTLoc to detect a contradiction). However, increasing the bound allows TACK to detect the counterexample.

When ae2Zot is used, the time required by TACK to verify models increases as the values of n and k increase, whereas the results obtained with ae2SBVZot are less homogeneous. Indeed, even if ae2SBVZot is in general more efficient than ae2Zot, some tests carried out by ae2SBVZot resulted in a timeout. For example, this is the case of property **live-three** with $k = 15$ and $n = 6$ which, conversely, has been successfully solved by ae2Zot. To understand the reason of this result and, in particular, whether it was caused by the adopted Zot plugin, two different versions of Z3 were compared with each other. A general variation of the performance of TACK—even when

Table 10. Time (s) required to check the properties of the Fischer benchmark. The symbol ✓ indicates that the property is satisfied, i.e., the CLTLoc formula is unsatisfiable. The symbol ✗ indicates that the property is not satisfied, i.e., the CLTLoc formula is satisfiable.

		TACK ae2zot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-one	10	0.9 ✓	1.2* ✓	0.9 ✓	1.1 ✓	1.4 ✓	1.4* ✓	1.9 ✓	2.2 ✓	2.0* ✓
	15	0.9 ✓	0.9* ✓	1.1* ✓	1.7 ✓	4.9 ✓	2.7 ✓	3.0* ✓	3.7* ✓	4.2* ✓
	20	1.0 ✓	1.2 ✓	1.6 ✓	2.5 ✓	3.0* ✓	5.1 ✓	4.4* ✓	78.7* ✓	12.8 ✓
	25	1.1 ✓	1.9 ✓	2.8 ✓	3.0* ✓	5.5 ✓	10.2 ✓	9.6* ✓	12.6* ✓	15.1* ✓
	30	1.3 ✓	15.6* ✓	3.1 ✓	4.0* ✓	6.5* ✓	11.1* ✓	20.5 ✓	17.8 ✓	22.4* ✓
live-two	10	1.8 ✓	1.8* ✓	2.6 ✓	2.9 ✓	2.6* ✓	4.7 ✓	5.5 ✓	5.1 ✓	4.3* ✓
	15	7.9 ✓	18.3 ✓	28.1 ✓	40.1 ✓	70.1 ✓	131.3 ✓	211.9 ✓	166.5* ✓	218.1* ✓
	20	42.1 ✓	96.5 ✓	139.9 ✓	514.8 ✓	897.1 ✓	1395.4 ✓	5837.2 ✓	–	–
	25	208.8 ✓	859.9 ✓	813.1 ✓	2026.9 ✓	6770.0 ✓	–	–	–	–
	30	392.7 ✓	793.5 ✓	2678.2 ✓	4193.4 ✓	–	–	–	–	–
live-three	10	1.1 ✗	1.6 ✗	1.4 ✗	1.7*✗	3.7* ✗	13.7* ✓	13.4* ✓	18.1* ✓	23.0* ✓
	15	1.2 ✗	1.6 ✗	2.4* ✗	4.3 ✗	15.0* ✗	42.3* ✗	33.5* ✗	143.7 ✗	100.4* ✗
	20	1.5 ✗	2.1* ✗	3.3* ✗	4.1* ✗	10.6* ✗	41.4 ✗	94.2 ✗	64.0* ✗	386.2* ✗
	25	2.3✗	3.4 ✗	5.0 ✗	11.0* ✗	20.4* ✗	15.4* ✗	84.5* ✗	354.0 ✗	648.9* ✗
	30	2.4 ✗	4.9✗	5.7* ✗	12.2 * ✗	30.1 ✗	37.6* ✗	124.5* ✗	532.3 ✗	155.8 ✗
live-four	10	1.4 ✓	1.4* ✓	2.2 ✓	2.5 ✓	2.9 ✓	3.5 ✓	2.7* ✓	3.4* ✓	5.9 ✓
	15	4.7 ✓	7.3 ✓	14.3 ✓	22.7 ✓	46.0 ✓	93.3 ✓	146.8 ✓	145.0* ✓	177.8* ✓
	20	10.4 ✓	24.4 ✓	54.5 ✓	98.5 ✓	199.9 ✓	797.2 ✓	2526.5 ✓	–	–
	25	48.5 ✓	112.9 ✓	191.0 ✓	779.3 ✓	1783.1 ✓	5437.316 ✓	–	–	–
	30	153.0 ✓	255.2 ✓	675.6 ✓	–	–	–	–	–	–
live-five	10	1.1 ✗	1.8 ✗	2.0 ✗	4.1 ✗	3.9* ✓	15.1* ✓	17.3* ✓	23.2* ✓	23.7* ✓
	15	1.5 ✗	1.9 ✗	3.8 ✗	9.1 ✗	16.3* ✗	20.6* ✗	42.2 ✗	24.7* ✗	182.7* ✗
	20	2.8 ✗	3.2 ✗	6.6 ✗	11.8* ✗	19.7* ✗	37.8 ✗	65.0* ✗	176.4* ✗	117.5* ✗
	25	3.5 ✗	4.1* ✗	5.0* ✗	18.1* ✗	28.2 ✗	54.6 ✗	123.1* ✗	420.2* ✗	794.4* ✗
	30	3.3* ✗	7.6 ✗	11.2* ✗	22.6 ✗	31.9* ✗	39.3* ✗	257.4 ✗	261.6* ✗	1940.4* ✗
live-six	10	0.9 ✓	1.8 ✓	1.7* ✓	2.8* ✓	5.0* ✓	6.3* ✓	11.1* ✓	13.0 ✓	21.9* ✓
	15	1.2 ✓	2.2* ✓	5.4* ✓	9.3* ✓	22.0* ✓	57.2* ✓	144.0* ✓	209.8* ✓	318.7* ✓
	20	1.3* ✓	4.7* ✓	16.7* ✓	51.6* ✓	146.5* ✓	350.6* ✓	857.6* ✓	1635.9* ✓	3734.3* ✓
	25	2.5 ✓	10.6* ✓	30.8* ✓	260.3* ✓	922.0* ✓	2388.1* ✓	3894.1*	6019.3*	–
	30	4.7 ✓	32.3 ✓	139.4* ✓	667.5* ✓	1906.6* ✓	5955.3*	–	–	–
		TACK ae2sbvzot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-one	10	0.7 ✓	0.7 ✓	0.8 ✓	0.9 ✓	0.9 ✓	1.0 ✓	2.4 ✓	1.1 ✓	1.3 ✓
	15	0.7 ✓	0.8 ✓	1.0 ✓	1.2 ✓	1.0 ✓	1.2 ✓	1.0* ✓	1.9 ✓	1.9 ✓
	20	0.7 ✓	0.8 ✓	1.0 ✓	1.4 ✓	1.5 ✓	2.1 ✓	2.0 ✓	2.1 ✓	2.2* ✓
	25	0.8 ✓	1.2 ✓	1.1* ✓	2.0* ✓	1.9 ✓	2.7 ✓	4.5 ✓	2.7* ✓	5.5* ✓
	30	0.9 ✓	1.2 ✓	1.7* ✓	1.7 ✓	2.1 ✓	2.2* ✓	2.2 ✓	3.6* ✓	4.8 ✓
live-two	10	1.2 ✓	1.3 ✓	1.7 ✓	1.5 ✓	1.6 ✓	1.5 ✓	1.7 ✓	1.9 ✓	2.3 ✓
	15	3.3 ✓	3.7 ✓	4.6 ✓	6.4 ✓	8.9* ✓	18.7 ✓	28.6 ✓	44.8 ✓	87.2 ✓
	20	5.5 ✓	13.8 ✓	19.0 ✓	45.0* ✓	54.3 ✓	60.2* ✓	127.5 ✓	60.2 ✓	1133.4* ✓
	25	16.2* ✓	17.9* ✓	51.6 ✓	93.5 ✓	33.1 ✓	136.9* ✓	592.0* ✓	3232.4* ✓	2359.7* ✓
	30	33.5* ✓	24.1 ✓	136.9 ✓	50.7 ✓	82.8* ✓	463.8* ✓	1750.6*✓	1927.7*✓	–
live-three	10	0.8 ✗	1.0 ✗	1.1 ✗	1.8* ✗	2.9* ✗	7.7 ✓	11.8* ✓	14.9 ✓	16.9 ✓
	15	1.4 ✗	1.1 ✗	1.2* ✗	2.0 ✗	13.5*✗	17.8* ✗	15.1* ✗	17.4 ✗	23.5 ✗
	20	1.1 ✗	1.3* ✗	1.9 ✗	3.5 ✗	6.1 ✗	4.5 ✗	6.7 ✗	51.6 ✗	87.1 ✗
	25	1.1 ✗	1.8 ✗	3.9 ✗	2.8 ✗	11.8 ✗	23.8 ✗	520.6 ✗	388.7 ✗	241.6* ✗
	30	1.3 ✗	2.2 ✗	2.9* ✗	16.8 ✗	10.3 ✗	30.8* ✗	126.6 ✗	71.4* ✗	142.2* ✗
live-four	10	0.9 ✓	1.0 ✓	1.2 ✓	1.3 ✓	1.4 ✓	1.6 ✓	1.5 ✓	2.1 ✓	1.8 ✓
	15	2.2 ✓	2.2* ✓	3.2 ✓	4.2* ✓	8.9 ✓	12.1* ✓	28.6* ✓	45.2 ✓	82.3 ✓
	20	4.0 ✓	6.6 ✓	9.0 ✓	8.0* ✓	35.8 ✓	69.6 ✓	19.9 ✓	345.6* ✓	1071.0* ✓
	25	11.0 ✓	16.6 ✓	22.1 ✓	17.2* ✓	31.2* ✓	474.6 ✓	2241.1* ✓	329.4 ✓	335.3*✓
	30	16.1 ✓	28.7* ✓	17.9* ✓	30.7* ✓	137.1 ✓	252.2* ✓	80.3*✓	1611.8 ✓	598.9*✓
live-five	10	0.9 ✗	1.3 ✗	1.9* ✗	1.3*✓	3.2*✓	17.9 ✓	16.5 ✓	17.9 ✓	13.4* ✓
	15	1.0 ✗	1.3 ✗	2.7 ✗	2.3 ✗	7.6*✗	19.4 ✗	19.2* ✗	26.6 ✗	26.6*✗
	20	1.6* ✗	1.7 ✗	3.6* ✗	4.7* ✗	4.8 ✗	16.8*✗	36.2 ✗	179.5*✗	36.9 ✗
	25	1.7 ✗	4.0 ✗	5.0* ✗	6.1 ✗	6.4* ✗	7.2 ✗	263.6* ✗	378.8*✗	308.5 ✗
	30	1.5 ✗	3.9 ✗	4.7 ✗	6.0* ✗	24.1 ✗	27.1* ✗	53.8 ✗	57.3 ✗	–
live-six	10	0.9 ✓	1.0 ✓	1.0* ✓	2.6 ✓	3.9* ✓	2.4* ✓	6.4* ✓	12.1 ✓	11.3* ✓
	15	0.8 ✓	1.6 ✓	3.5 ✓	5.2 ✓	21.1 ✓	39.9 ✓	65.5 ✓	112.3 ✓	205.7 ✓
	20	1.2 ✓	3.5 ✓	6.9 ✓	17.3✓	40.5* ✓	170.6 ✓	278.3* ✓	650.8* ✓	2555.4 ✓
	25	1.5 ✓	4.8* ✓	19.5* ✓	75.2 ✓	110.3* ✓	672.1 ✓	1485.4* ✓	3596.3*✓	5861.6*✓
	30	1.9 ✓	11.7 ✓	35.1 ✓	116.3* ✓	506.6* ✓	2522.8* ✓	3945.9*✓	–	–

using the same plugin—is evident, and it stems from the tactics that are used by Z3 to solve the satisfiability problem. In fact, the experiments using version 4.4.1 of Z3 show that TACK is able to complete the verification for the cases that resulted in timeouts using version 4.7.1, though it timed out in others. This evidence proves that the choice of the Zot plugin does not determine the presence or absence of the timeouts. In Table 10, the cases in which version 4.4.1 of Z3 significantly outperformed version 4.7.1 (by at least around 25%) are marked with the * symbol.

CSMA/CD Protocol. The CSMA/CD protocol (Carrier Sense, Multiple-Access with Collision Detection) aims at assigning a bus to one of n competing stations. When a station has data to send, it first listens to the bus. If no other station is transmitting (the bus is idle), the station begins the transmission. If another station is transmitting (the bus is busy), it waits a random amount of time and then repeats the previous steps. The synchronization among the participants is obtained through a broadcast transition-based synchronization.

The following MITL property was tested. It is inspired by the one considered in the Uppaal benchmark [26].

$$\textbf{live-csma} := \mathcal{G}_{[0,\infty)}(P_1.start\_send \rightarrow (\neg collision\_after\_transm)) \tag{11}$$

$$P_1.start\_send := (\neg P_1.send) \wedge (P_1.send\, \mathcal{U}_{(0,inf)} \top)) \tag{12}$$

$$collision\_after\_transm := \mathcal{G}_{(0,52]}(P_1.send \wedge (P_1.send\, \mathcal{U}_{[0,inf)}(P_1.send \wedge P_2.send))) \tag{13}$$

The property **live-csma** predicates on the occurrence of a collision—i.e., P_1 and P_2 simultaneously sending a message. It specifies that a collision does not occur after P_1 is transmitting for 52 time units or more. Let us consider formula $P_1.start\_send$ (12). Formula $\neg P_1.send$ specifies that P_1 is not sending at the current time t. Formula $P_1.send\, \mathcal{U}_{[(0,\infty)} \top$ specifies that there exists a t' such that, for every t'' s.t. $t < t'' < t'$ holds, $P_1.send$ holds. Thus, formula $P_1.start\_send$ is true when P_1 starts sending a message. Let us now consider formula $collision\_after\_transm$ (13), which specifies that P_1 transmits for 52 time units or more, and then a collision is detected. Operator $\mathcal{G}_{(0,52]}$ forces formula $(P_1.send) \wedge (P_1.send\, \mathcal{U}_{[0,\infty)}(P_1.send \wedge P_2.send)))$ to hold continuously from the current time instant, until 52 time units from now, included. Since the formula must also hold at time instant 52 from now, it forces a collision to be detected at a time instant that is after 52 time units from now. Furthermore, since the formula must hold in interval $(0, 52]$, P_1 must keep sending a message within this interval. The results of the experiments are presented in Table 11.

Token Ring. The token ring benchmark considers n symmetric stations that are organized in a ring, plus one process that models the ring. The ring moves the token on a given direction among the n processes. The processes may hand back the token in a synchronous (high-speed) or an asynchronous (low priority) fashion. The synchronization among the participants is obtained through a channel transition-based synchronization.

The following MITL property was tested. It is inspired by—though it is not the same as—the one considered in the Uppaal benchmark [26].

$$\begin{aligned}\textbf{live-token} := \; & \mathcal{G}_{(0,\infty)}(\neg ((ST_1.zsync \vee ST_1.zasync \vee ST_1.ysync \vee ST_1.yasync) \wedge \\ & (ST_2.zsync \vee ST_2.zasync \vee ST_2.ysync \vee ST_2.yasync)))\end{aligned}$$

Property **live-token** specifies that two stations ST_1 and ST_2 can not simultaneously sync—i.e., while one of them is in a synchronize state the other must be idle. The results are presented in Table 12.

The results presented in Tables 10, 11 and 12 show that in all the cases the verification time is reasonable for practical adoptions of the proposed verification technique. Furthermore, the proposed technique easily allows considering different semantics—e.g., different synchronization mechanisms—without directly changing the verification algorithm.

Table 11. Time (s) required to check the property of the CSMA/CD Protocol. The symbol ✓ indicates that the property is satisfied, i.e., the CLTLoc formula is unsatisfiable. The symbol ✗ indicates that the property is not satisfied, i.e., the CLTLoc formula is satisfiable.

		TACK ae2zot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-csma	10	2.6 ✓	5.4 ✓	5.8 ✓	7.1 ✓	9.9 ✓	7.6 ✓	11.3 ✓	12.3 ✓	16.0 ✓
	15	10.5 ✓	19.5 ✓	23.1 ✓	45.8 ✓	64.0 ✓	135.4 ✓	123.9 ✓	221.5 ✓	453.3 ✓
	20	20.8 ✓	46.5 ✓	97.3 ✓	140.8 ✓	409.9 ✓	663.5 ✓	1146.6 ✓	1102.9 ✓	1299.4 ✓
	25	81.2 ✓	125.4 ✓	220.3 ✓	387.9 ✓	1278.6 ✓	1959.1 ✓	4742.7 ✓	2820.2 ✓	7184.4 ✓
	30	98.8 ✓	389.4 ✓	868.0 ✓	1500.9 ✓	2195.1 ✓	-	-	-	-
		TACK ae2sbvzot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-csma	10	1.6 ✓	1.7 ✓	2.1 ✓	2.2 ✓	2.9 ✓	2.8 ✓	3.3 ✓	3.8 ✓	3.9 ✓
	15	4.4 ✓	6.9 ✓	8.8 ✓	7.6 ✓	8.4 ✓	24.3 ✓	32.9 ✓	24.2 ✓	21.1 ✓
	20	9.0 ✓	15.5 ✓	12.0 ✓	26.2 ✓	32.3 ✓	35.2 ✓	49.4 ✓	75.6 ✓	65.0 ✓
	25	19.2 ✓	21.7 ✓	45.3 ✓	68.9 ✓	107.4 ✓	143.6 ✓	178.5 ✓	267.2 ✓	245.7 ✓
	30	70.3 ✓	68.7 ✓	151.6 ✓	130.0 ✓	438.9 ✓	328.9 ✓	4441.8 ✓	854.2 ✓	6649.4 ✓

Table 12. Time (s) required to check the property of the Token Ring. The symbol ✓ indicates that the property is satisfied, i.e., the CLTLoc formula is unsatisfiable. The symbol ✗ indicates that the property is not satisfied, i.e., the CLTLoc formula is satisfiable.

		TACK ae2zot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-token	10	0.9 ✓	1.1 ✓	1.3 ✓	2.1 ✓	1.9 ✓	2.1 ✓	2.1 ✓	2.3 ✓	2.2 ✓
	15	1.5 ✓	1.5 ✓	1.8 ✓	2.2 ✓	3.9 ✓	4.8 ✓	3.7 ✓	3.2 ✓	9.0 ✓
	20	2.2 ✓	2.2 ✓	4.8 ✓	3.1 ✓	5.0 ✓	10.6 ✓	7.1 ✓	18.9 ✓	10.4 ✓
	25	2.7 ✓	5.0 ✓	3.7 ✓	5.8 ✓	5.7 ✓	24.3 ✓	25.6 ✓	19.6 ✓	58.2 ✓
	30	6.0 ✓	9.9 ✓	6.9 ✓	17.6 ✓	27.3 ✓	36.3 ✓	43.8 ✓	21.3 ✓	36.0 ✓
		TACK ae2sbvzot								
		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-token	10	0.9 ✓	0.9 ✓	0.9 ✓	1.0 ✓	1.1 ✓	1.2 ✓	1.3 ✓	1.5 ✓	1.6 ✓
	15	1.2 ✓	1.1 ✓	1.1 ✓	1.1 ✓	1.2 ✓	1.4 ✓	1.5 ✓	1.7 ✓	1.7 ✓
	20	2.1 ✓	1.9 ✓	1.8 ✓	1.6 ✓	1.8 ✓	1.8 ✓	1.8 ✓	2.1 ✓	2.2 ✓
	25	2.5 ✓	3.7 ✓	3.5 ✓	3.1 ✓	2.3 ✓	2.4 ✓	2.5 ✓	2.9 ✓	2.9 ✓
	30	3.6 ✓	5.6 ✓	4.8 ✓	5.3 ✓	4.1 ✓	3.5 ✓	3.0 ✓	3.2 ✓	3.9 ✓

As mentioned above, an exhaustive, direct comparison of TACK with $\text{MITL}_{0,\infty}$BMC and MightyL over the considered benchmarks is not possible. For example, both the models of CSMA/CD and Token Ring protocols rely on synchronization primitives that are not supported by $\text{MITL}_{0,\infty}$BMC and MightyL. However, a model of Fischer protocol, which does not need synchronization primitives to be used, is available in the $\text{MITL}_{0,\infty}$BMC distribution, so it is at least possible to perform a

Table 13. Time (s) required to check the properties of the Fischer benchmark using the $\text{MITL}_{0,\infty}$BMC tool. The symbol ✓ indicates that the property is satisfied, i.e., the formula input to the underlying solver is unsatisfiable. The symbol ✗ indicates that the property is not satisfied, i.e., the formula input to the underlying solver is satisfiable.

		n								
	k	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
live-one	10	0.4 ✓	0.4 ✓	0.5 ✓	0.7 ✓	0.8 ✓	0.9 ✓	1.1 ✓	1.3 ✓	1.6 ✓
	15	0.9 ✓	1.0 ✓	1.2 ✓	1.5 ✓	1.8 ✓	2.4 ✓	2.7 ✓	3.0 ✓	3.5 ✓
	20	2.0 ✓	2.3 ✓	2.4 ✓	3.0 ✓	3.4 ✓	4.3 ✓	5.2 ✓	5.7 ✓	6.4 ✓
	25	4.4 ✓	4.1 ✓	4.4 ✓	4.9 ✓	5.7 ✓	7.3 ✓	8.5 ✓	9.7 ✓	10.7 ✓
	30	8.2 ✓	7.2 ✓	6.8 ✓	10.7 ✓	13.3 ✓	11.2 ✓	12.7 ✓	20.8 ✓	16.3 ✓
live-two	10	0.6 ✓	0.9 ✓	1.0 ✓	1.1 ✓	1.6 ✓	1.7 ✓	2.0 ✓	2.1 ✓	3.6 ✓
	15	3.1 ✓	6.9 ✓	16.4 ✓	13.4 ✓	25.3 ✓	11.3 ✓	19.5 ✓	20.8 ✓	91.0 ✓
	20	27.9 ✓	43.6 ✓	65.6 ✓	123.4 ✓	181.8 ✓	138.6 ✓	210.5 ✓	233.5 ✓	458.2 ✓
	25	168.2 ✓	120.8 ✓	163.1 ✓	271.1 ✓	346.1 ✓	294.6 ✓	626.5 ✓	358.6 ✓	935.5 ✓
	30	462.1 ✓	270.8 ✓	307.3 ✓	444.5 ✓	609.7 ✓	587.7 ✓	1229.1 ✓	714.7 ✓	1491.1 ✓
live-three	10	0.2 ✗	0.3 ✗	0.4 ✗	0.4✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	15	0.2 ✗	0.3 ✗	0.4 ✗	0.4✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	20	0.2 ✗	0.3 ✗	0.4 ✗	0.4✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	25	0.2 ✗	0.3 ✗	0.4 ✗	0.4✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	30	0.2 ✗	0.3 ✗	0.4 ✗	0.4✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
live-four	10	0.5 ✓	0.9 ✓	1.0 ✓	1.5 ✓	1.6 ✓	1.6 ✓	2.4 ✓	2.6 ✓	2.4 ✓
	15	3.3 ✓	7.2 ✓	8.6 ✓	12.9 ✓	43.4 ✓	13.9 ✓	73.3 ✓	46.0 ✓	77.7 ✓
	20	23.8 ✓	63.5 ✓	54.2 ✓	82.1 ✓	146.7 ✓	140.5 ✓	116.4 ✓	189.0 ✓	215.7 ✓
	25	107.3 ✓	159.3 ✓	142.8 ✓	246.1 ✓	293.3 ✓	245.5 ✓	380.7 ✓	300.7✓	663.2 ✓
	30	367.8 ✓	372.6 ✓	444.5 ✓	707.7 ✓	572.6 ✓	658.3 ✓	1156.7 ✓	1457.8 ✓	1351.1 ✓
live-five	10	0.2 ✗	0.3 ✗	0.4 ✗	0.5 ✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	15	0.2 ✗	0.3 ✗	0.4 ✗	0.5 ✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	20	0.2 ✗	0.3 ✗	0.4 ✗	0.5 ✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	25	0.2 ✗	0.3 ✗	0.4 ✗	0.5 ✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
	30	0.2 ✗	0.3 ✗	0.4 ✗	0.5 ✗	0.5 ✗	0.6 ✗	0.7 ✗	0.8 ✗	0.9 ✗
live-six	10	0.2 ✓	0.4 ✓	0.6 ✓	0.8 ✓	0.9 ✓	1.3 ✓	1.5 ✓	1.7 ✓	2.0 ✓
	15	0.7 ✓	2.5 ✓	6.1 ✓	9.7 ✓	13.9 ✓	21.5 ✓	26.2 ✓	35.1 ✓	52.1 ✓
	20	2.5 ✓	19.7 ✓	57.5 ✓	93.4 ✓	151.7 ✓	239.8 ✓	350.7 ✓	548.9 ✓	738.9 ✓
	25	7.9 ✓	123.5 ✓	364.2 ✓	597.9 ✓	1177.0 ✓	1985.5 ✓	4001.2 ✓	6956.2 ✓	–
	30	20.2 ✓	538.0 ✓	1764.0 ✓	3651.1 ✓	–	–	–	–	–

Table 14. Time (s) required by TACK (for different bounds k) to model check the timed lamp [11], and the results obtained by Brihaye et al. [15] when MightyL is combined with LTSMIN to verify this example.

	MightyL + LTSMIN [15]		TACK				
			k				
	with minimality	w/o minimality	10	15	20	25	30
φ_1	1.73	1.77	1.3 ✗	1.5 ✗	2.4 ✗	2.6 ✗	2.8 ✗
φ_2	2.36	13.18	1.4 ✓	1.6 ✓	1.7 ✓	2.3 ✓	3.1 ✓

set of verification experiments with it. In addition, properties **live-one** to **live-six** are $\text{MITL}_{0,\infty}$ formulae, so they can be verified by $\text{MITL}_{0,\infty}$BMC without modification. Nevertheless, the semantic discrepancies between TACK and $\text{MITL}_{0,\infty}$BMC highlighted above still remain, so even if in the case of the Fischer protocol a comparison between the two tools has some merit, it is still not fully meaningful. Despite these issues, we verified properties **live-one** to **live-six** for the Fischer protocol, using $\textsc{Mitl}_{0,\infty}$BMC with the same experimental setup (parameters and hardware configuration) as those used for TACK to obtain the results of Table 10. Table 13 shows the execution times obtained. It can be noticed that, when the properties hold for the model (✓, which means that the formula analyzed by the tool is unsatisfiable), in most cases TACK is faster. When the properties do not hold, though (✗, which means that the formula analyzed is satisfiable), the incremental fashion that $\textsc{Mitl}_{0,\infty}$BMC uses to explore the state space proves to be very beneficial. Indeed, an analysis of the trace returned by the tool in these cases shows that after reaching bound $k = 6$ the tool determines that the formula is satisfiable, so it stops the exploration (in fact, the execution times are independent of the bound, which is always greater than 6). Notice also that, for properties **live-three** and **live-five**, a bound of 10 may not be enough for TACK to find the counterexample, whereas it is for $\textsc{Mitl}_{0,\infty}$BMC; this further highlights that the two tools, being based on slightly different semantics, are not entirely comparable, even given the similarity in their approaches.

Finally, even though, for the reasons outlined above, we could not run MightyL with the same configuration as the one used for TACK, we used the results presented in [15] on a small example, namely the timed lamp [11], over two different properties φ_1 and φ_2. Table 14 reports the results presented in [15] and those obtained using TACK (with ae2zot and Z3 4.7.1) for different bound values k. The results are not meant to be compared directly, but provide a qualitative comparison among the tools, and further show the viability of our approach.

7 CONCLUSION

This paper presented a flexible approach for checking networks of TA against properties expressed in MITL. The technique relies on an intermediate artifact—i.e., a CLTLoc formula—in which both the model and the property are encoded. The intermediate artifact is then evaluated using suitable satisfiability checkers. The proposed technique addresses three main challenges: (i) it allows considering a signal-based semantics; (ii) it allows verifying properties expressed using the MITL; (iii) it allows easily adding new TA constructs and changing their semantics (e.g., synchronization mechanisms, liveness conditions and edge constraints).

The technique has been implemented in an open source tool called TACK (Timed Automata ChecKer), which is publicly available at http://github.com/claudiomenghi/TACK. Evaluation is performed by assessing: (i) the efficiency of TACK in verifying MITL properties of TA; (ii) the possibility of considering different synchronization constructs and semantic constraints. The intermediate artifact is evaluated through a bounded model checking technique that relies on two different solvers available in the Zot formal verification tool [6]. Evaluation mainly relies on three different benchmarks that have been used to evaluate similar artifacts (e.g., [2]), namely the Fischer mutual exclusion protocol [3], the CSMA/CD protocol [1] and the Token Ring protocol [23]. The results show that the verification time is reasonable for practical adoptions of the proposed verification technique and prove that the proposed technique easily allows considering different semantics—e.g., different synchronization mechanisms—by simply adding and removing formulae in the intermediate CLTLoc encoding.

ACKNOWLEDGEMENTS

This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277).
We are grateful for the feedback provided by the anonymous reviewers.

REFERENCES

[1] [n. d.]. IEEE Web Page. http://www.ieee802.org/3

[2] [n. d.]. Uppaal Web Page. http://www.uppaal.org/
[3] Martín Abadi and Leslie Lamport. 1994. An old-fashioned recipe for real time. *Transactions on Programming Languages and Systems* (1994), 1543–1571.
[4] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.
[5] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. 1996. The benefits of relaxing punctuality. *Journal of the ACM (JACM)* 43, 1 (1996), 116–146.
[6] Luciano Baresi, Mohammad Mehdi Pourhashem Kallehbasti, and Matteo Rossi. 2016. How Bit-vector Logic Can Help Improve the Verification of LTL Specifications over Infinite Domains. In *Annual ACM Symposium on Applied Computing (SAC)*. ACM.
[7] Marcello M. Bersani, Carlo A. Furia, Matteo Pradella, and Matteo Rossi. 2009. Integrated Modeling and Verification of Real-Time Systems through Multiple Paradigms. In *Software Engineering and Formal Methods, SEFM*. IEEE, 13–22.
[8] Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. 2013. Deciding the Satisfiability of MITL Specifications. In *Proc. of the Int. Symp. on Games, Automata, Logics and Formal Verification (GandALF)*. 64–78.
[9] Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. 2015. An SMT-based approach to Satisfiability Checking of MITL. *Inform. and Comp.* 245 (2015), 72–97.
[10] Marcello M Bersani, Matteo Rossi, and Pierluigi San Pietro. 2016. A logical characterization of timed regular languages. *Theoretical Computer Science* (2016).
[11] Marcello M Bersani, Matteo Rossi, and Pierluigi San Pietro. 2016. A tool for deciding the satisfiability of continuous-time metric temporal logic. *Acta Informatica* 53, 2 (2016), 171–206.
[12] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. 2010. On the expressiveness of TPTL and MTL. *Information and Computation* 208, 2 (2010), 97–116.
[13] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, Arthur Milchior, and Benjamin Monmege. 2018. Efficient algorithms and tools for MITL model-checking and synthesis. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 180–184.
[14] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. 2017. MightyL: A Compositional Translation from MITL to Timed Automata. In *International Conference on Computer Aided Verification*. Springer, 421–440.
[15] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. 2017. Timed-Automata-Based Verification of MITL over Signals. In *Symposium on Temporal Representation and Reasoning (TIME) (Leibniz International Proceedings in Informatics (LIPIcs))*, Sven Schewe, Thomas Schneider, and Jef Wijsen (Eds.), Vol. 90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
[16] Alessandro Carioni, Silvio Ghilardi, and Silvio Ranise. 2010. MCMT in the Land of Parametrized Timed Automata. In *International Verification Workshop (VERIFY)*. EasyChair, 47–64.
[17] Zhe Dang. 2003. Pushdown timed automata: a binary reachability characterization and safety verification. *Theoretical Computer Science* 302, 1 (2003), 93 – 121. https://doi.org/10.1016/S0304-3975(02)00743-0
[18] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.
[19] Stéphane Demri and Deepak D'Souza. 2007. An automata-theoretic approach to constraint LTL. *Information and Computation* 205, 3 (2007), 380–415.
[20] Deepak D'Souza and Pavithra Prabhakar. 2007. On the expressiveness of MTL in the pointwise and continuous semantics. *International Journal on Software Tools for Technology Transfer (STTT)* 9, 1 (2007), 1–4.
[21] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. 2012. *Modeling Time in Computing*. Springer.
[22] Yoram Hirshfeld and Alexander Moshe Rabinovich. 2004. Logics for Real Time: Decidability and Complexity. *Fundamenta Informaticae* 62, 1 (2004), 1–28.
[23] Raj Jain. 1994. *FDDI handbook: high-speed networking using fiber and other media*. Addison-Wesley Longman Publishing Co., Inc.
[24] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. 2015. LTSmin: high-performance language-independent model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.
[25] Roland Kindermann, Tommi Junttila, and Ilkka Niemelä. 2013. Bounded model checking of an MITL fragment for timed automata. In *International Conference on Application of Concurrency to System Design (ACSD)*. IEEE, 216–225.
[26] Kim G Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer* 1, 1 (1997), 134–152.
[27] Didier Lime, Olivier H Roux, Charlotte Seidner, and Louis-Marie Traonouez. 2009. Romeo: A parametric model-checker for Petri nets with stopwatches. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.
[28] Oded Maler, Dejan Nickovic, and Amir Pnueli. 2006. From MITL to Timed Automata. In *Proc. of FORMATS*. LNCS, Vol. 4202. 274–289.

[29] Francesco Marconi, Marcello M Bersani, Madalina Erascu, and Matteo Rossi. 2016. Towards the Formal Verification of Data-Intensive Applications Through Metric Temporal Logic. In *International Conference on Formal Engineering Methods*. Springer.

[30] P. M. Merlin. [n. d.]. *A study of the recoverability of computing systems*. PhD thesis. University of California - Irvine, CA.

[31] Joël Ouaknine and James Worrell. 2008. Some Recent Results in Metric Temporal Logic. In *FORMATS (LNCS)*, Vol. 5215. Springer, 1–13.

[32] Farn Wang. 2001. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram. In *International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*. Kluwer, B.V.

[33] Sergio Yovine. 1997. Kronos: A Verification Tool for Real-Time Systems. (Kronos User's Manual Release 2.2). *International Journal on Software Tools for Technology Transfer* 1 (1997), 123–133.

A SIMPLIFIED ENCODING

The encoding used in the experiments is simpler than the general one presented in Fig. 4 of Sec. 4 because it is tailored only to signals whose intervals are all left-open and right-closed. In such a case, the encoding can be simplified as the distinction between the kind of transitions is no longer needed.

A.1 Encoding of traces for left-open right-closed signals.

The following Fig. 8 shows the simplified encoding of the network traces which, however, still retains the structure of the general one. In particular, the atom edge$^{](}$ becomes irrelevant, and then it can be removed, and the formulae φ_4 and φ_5 are modified.

$$\varphi_1 := \bigwedge_{k\in[1,K]} (\mathtt{l}[k] = 0) \qquad \varphi_2 := \bigwedge_{n\in Int} n = v^0_{\mathrm{var}}(n) \qquad \varphi_3 := \bigwedge_{k\in[1,K]} Inv(\mathtt{l}[k])$$

$$\varphi_4 := \bigwedge_{\substack{k\in[1,K]\\ q\in Q_k}} ((\mathtt{l}[k] = q \wedge \mathtt{t}[k] = \natural) \rightarrow \mathcal{X}(r_1(Inv(q)))$$

$$\varphi_5 := \bigwedge_{k\in[1,K],\, t\in T_k} \mathtt{t}[k] = t \rightarrow \left(\mathtt{l}[k] = t^- \wedge \phi_\xi \wedge \mathcal{X}(\mathtt{l}[k] = t^+ \wedge \phi_\gamma \wedge \phi_\mu \wedge \phi_\zeta \wedge \phi_{\alpha^{](}}(t^-, t^+, k)\right)$$

$$\phi_{\alpha^{](}}(a, b, i) := Inv(a) \wedge r_2(Inv_w(b))$$

$$\varphi_6 := \bigwedge_{k\in[1,K],\, q,q'\in Q_k \mid q\neq q'} \left(((\mathtt{l}[k] = q) \wedge \mathcal{X}(\mathtt{l}[k] = q')) \rightarrow \bigvee_{t\in T_k,\, t^-=q,\, t^+=q'} (\mathtt{t}[k] = t)\right)$$

$$\varphi_7 := \bigwedge_{x\in X} \left(\mathcal{X}(x_0 = 0 \vee x_1 = 0) \rightarrow \bigvee_{\substack{k\in[1,K]\\ t\in T_k \mid x\in t_\zeta}} \mathtt{t}[k] = t\right) \qquad \varphi_8 := \bigwedge_{n\in Int} \left((\neg(n = \mathcal{X}(n))) \rightarrow \bigvee_{\substack{k\in[1,K]\\ t\in T_k \mid n\in U(t)}} \mathtt{t}[k] = t\right)$$

Fig. 8. Encoding of the automaton.

A.2 Encoding of left-open right-closed signals.

The following Fig. 9 shows the simplified encoding of the signals. The formulae φ_1 and φ_2 are the same as those in the general encoding in Fig. 9. The definition of the signal in every interval determined by the trace is simpler because it does not depend anymore on the kind of transition performed by the automata. For instance, if automaton k is in $\mathtt{l}[k]$ at position h then the over the interval I_h and in its right end-point the atomic propositions in the signal include those associated with $\mathtt{l}[k]$. A similar argument holds for the value of integer values.

$\mu_1 \coloneqq \mathcal{G} \bigwedge_{a \in AP} \overleftarrow{a} \leftrightarrow \bigvee_{k \in (0,K], q \in Q_k, a \in L(q)} (1[k] = q)$	$\mu_2 \coloneqq \mathcal{G} \bigwedge_{(n \sim d) \in AF} (\overleftarrow{n \sim d} \leftrightarrow n \sim d)$
$\mu_3 \coloneqq \bigwedge_{\substack{k \in (0,K],\\ a \in AP}} ⫯_a \leftrightarrow \bigvee_{a \in L(q_{0,k})} l[k] = q_{0,k}$	$\mu_4 \coloneqq \bigwedge_{(n \sim d) \in AF} \big(⫯_{n \sim d} \leftrightarrow n \sim d\big)$
$\mu_5 \coloneqq \bigwedge_{\substack{k \in (0,K],\\ a \in AP}} \overleftarrow{a} \rightarrow \mathcal{X}(⫯_a)$	$\mu_6 \coloneqq \bigwedge_{\substack{k \in (0,K],\\ (n \sim d \in AF)}} \overleftarrow{n \sim d} \rightarrow \mathcal{X}(⫯_{n \sim d})$

Fig. 9. Encoding of left-open right-closed signals.