From Real-Time Logic to Timed Automata*

THOMAS FERRÈRE, IST Austria
ODED MALER†, CNRS-Verimag, University of Grenoble-Alpes
DEJAN NIČKOVIĆ, AIT Austrian Institute of Technology
AMIR PNUELI‡, Weizmann Institute of Science and New York University

We show how to construct temporal testers for the logic MITL, a prominent linear-time logic for real-time systems. A temporal tester is a transducer which inputs a signal holding the Boolean value of atomic propositions, and outputs the truth value of a formula along time. Here we consider testers over continuous-time Boolean signals that use clock variables to enforce duration constraints, as in timed automata. We first rewrite the MITL formula into a "simple" formula using a limited set of temporal modalities. We then build testers for these specific modalities, and show how to compose testers for simple formulae into complex ones. Temporal testers can be turned into acceptors, yielding a compositional translation from MITL to timed automata. This construction is much simpler than previously known and remains asymptotically optimal. It supports both past and future operators and can easily be extended.

CCS Concepts: • **General and reference** → **Verification**; • **Theory of computation** → *Timed and hybrid models*; *Modal and temporal logics*;

ACM Reference Format:
Thomas Ferrère, Oded Maler, Dejan Ničković, and Amir Pnueli. 2018. From Real-Time Logic to Timed Automata. *J. ACM* 1, 1 (November 2018), 31 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

This paper presents a novel compositional translation from a real-time specification formalism, *metric interval temporal logic* (MITL) [5] with both past and future operators, to timed automata [4]. Such a translation is a crucial ingredient in the application of formal and semi-formal system validation methodologies (model checking, model-based testing, runtime verification) to real-time, that is, to models that capture *quantitative* timing aspects of system behaviors. To put this work in context, we start by describing the role of such a translation in the classical "untimed" setting and summarize previous work on lifting it to the quantitative timed case.

Temporal logic [58, 59] is a commonly-used specification formalism for discrete transition systems. The algorithmic verification of such systems goes by the name *model checking*, because decision procedures that check whether a sequence, set of sequences, or system is a model of a temporal logic formula play a central role in the verification process [12, 14, 26–28, 43, 70]. In the linear-time context one takes the negation $\neg\varphi$ of the specification and derives from it an automaton-like device $\mathcal{A}_{\neg\varphi}$ that accepts exactly sequences of states and actions that violate φ [76] and then checks whether the set of behaviors generated by the system model intersects the language of $\mathcal{A}_{\neg\varphi}$.

*This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).
†February 21, 1957 – September 3, 2018.
‡April 22, 1941 – November 2, 2009.

0004-5411/2018/11-ART $15.00
https://doi.org/0000001.0000001

For discrete-time models, used for functional verification of software or synchronous hardware, the logical situation is rather mature. Logics such as LTL (linear-time temporal logic) or CTL (computation-tree logic) are commonly accepted and incorporated into verification tools. For LTL a variety of efficient algorithms for translating a formula into an equivalent automaton have been proposed [33, 34, 44, 73] and underly industrial standards [31] such as PSL [1] and SVA [2].

When a temporal logic such as LTL is used in practice, one usually considers only its *future* fragment, where the temporal modalities refer to future occurences of events. It has been argued that such a futuristic specification style is more natural for humans, and this approach has been indeed adopted by aforementioned industrial specification languages PSL and SVA. Moreover, the *past* fragment of LTL does not add any expressive power to its future fragment, when interpreted over sequences that have a starting point.[1]

However, combining both past and future LTL operators may still be advantageous by allowing to express certain properties more succinctly [60] and more naturally. To see the latter point consider first the very typical property "every p is followed by a q" stating, for example, the fact that every request is eventually granted. This property is naturally expressed by the future property $\Box(p \rightarrow \Diamond\, q)$. On the other hand the dual property "every q should have been preceded by a p" forbidding unsolicited grants is expressed naturally using past operators as $\Box(q \rightarrow \diamondminus\, p)$ while its formulation in pure future LTL is cumbersome. Another property which makes use of past operators and whose realization in dense time will be discussed in the sequel, is $\uparrow p$ (read "rising edge of p") which holds at time instants where p becomes true. This property is naturally expressed as $p \wedge \neg \ominus p$, which says literally "p and not previously p". A more exhaustive list of mixed future-past properties can be found in [49].

When considering *timed* models and specification formalisms whose semantics involves the time domain $\mathbb{R}_{\geq 0}$ rather than $\mathbb{N}$, the situation is somewhat less satisfactory [7, 9, 36, 74]. Many variants of real-time logics [5, 8, 13, 37, 39, 48, 71, 77] as well as timed regular expressions [10, 11] have been proposed but the correspondence between simply-defined logics and variants of timed automata (automata with auxiliary clock variables [4]) is not as simple and canonical as for the untimed case, partly, of course, due to the additional complexity of the timed model (see also [56]). Consequently, existing verification tools for timed automata rarely use rich temporal properties.

One of the most popular dense-time extensions of LTL is the logic MITL (metric interval temporal logic) introduced in [5] as a fragment of the logic MTL [48] reinterpreted over continuous time. The semantic objects that satisfy or violate MITL formulae are *Boolean signals*, functions from $\mathbb{R}_{\geq 0}$ to valuations of the propositional variables appearing in the formula. The principal modality of MITL is the *timed until* denoted $\mathcal{U}_I$ and parameterized by nonsingular interval I with integer endpoints. Formula $p\,\mathcal{U}_{[a,b]}\,q$ is satisfied at any time instant t by a signal that admits q at some $t' \in [t+a, t+b]$, and where p holds continuously from t to t'. The restriction of time modalities to positive-length intervals was intended to guarantee decidability although later, unexpected results [66] showed that this restriction is not necessary for deciding MTL over finitary event-based semantics. The original version of MITL contained only future temporal operators and [5] give a procedure for translating an MITL formula into a nondeterministic timed automaton with the satisfiability and model-checking problems being EXPSPACE-complete. An investigation of past and future versions of MITL was carried out in [6] using *two-way* timed automata. Several variants of MITL with finite automata [37, 77] and threshold counting [42] have been studied. We also mention the connections of MITL and its extensions with monadic logic [36, 41, 77].

[1]The suggestion to use this "anchored" model of time for temporal logic was first advocated in [57]. It makes the time domain isomorphic to $\mathbb{N}$, rather than $\mathbb{Z}$. Languages over bi-infinite sequences have been studied in [65].

The problem of model checking real-time systems has been reviewed in [16] and we briefly survey some related work on MITL. In [66], against a previously held belief the satisfiability of future-only MTL was shown decidable over the semantic model of timed words. The decision procedure in [66] involves translating MTL formulae to one-clock alternating timed automata. Based on this result, [17, 18] describes a novel translation from MITL to timed automata using a form of *dealternation* applicable to the alternating timed automata translating MITL. The resulting implementation [19] can be interfaced with the timed automaton model checker Uppaal [51]. Using an equisatisfiable reduction, this allows to also decide the satisfiability of MITL over the semantic model of continuous-time Boolean signals [20]. Satisfiability-preserving translations from MITL to LTL were also proposed, using a *timer* normal form [40] or through the introduction of clock variables in the logic [15], the latter reduction enjoying a concrete implementation. In [46], the authors propose a super-dense time semantics for the $\mathsf{MITL}_{0,\infty}$ fragment of the logic, devise an encoding of both specifications and timed automata models to symbolic transition systems and develop a bounded model checking algorithm. A compositional translation from MITL with past to signal automata is proposed in [47]. This construction relies on the rewriting of the original formula to another equisatisfiable formula that uses only past operators. Consequently, this translation can be done only for specifications with bounded future modalities. In [72], some steps of the original translation of MITL to timed automata [5] have been formalized using the proof system PVS [67], correcting an error in the original semantics of the timed *release* operator.[2]

In our opinion, the tableau-based automaton construction in [5] remains unintuitive and rather complicated. Preliminary versions of this paper [53, 54] introduced a much simpler alternative to the construction of [5]. The modular translation from future MITL to timed automata in [54] is based on the concept of timed *temporal testers*, which were already applied in the discrete-time context to LTL [45, 62] and CTL* [44]. In this framework, one associates a nondeterminstic (timed) transducer with every sub-formula according to its temporal operator. At any time instant t, the output of the transducer represents the satisfiability of that sub-formula at t. A network of testers whose structure conforms to the parse tree of the formula recognizes the timed language defined by the formula. The procedure is conceptually simpler than the ones commonly used to model-check LTL based on alternating automata or tableaux, which often rely on generalized Büchi automata as an intermediate step [33, 35]. The temporal tester directly provides such an automaton.

Regarding MITL, part of the simplicity in the construction of [54] was a result of a restriction imposed on the considered signals and, consequently, on the syntax and semantics of the temporal MITL operators which slightly deviated from those of [5]. First, models were restricted to signals for which the time domain can be covered by *left-closed right-open* intervals on each of which the signal value remains constant. This restriction forbids signals in which a value can hold in isolated "singular" time points. To guarantee the closure of this property under temporal operators, intervals in the time modalities were then restricted to be *closed* of the form $[a, b]$. A slight modification in the semantics of the *until* operator was also considered, insisting that in order for $p \,\mathcal{U}\, q$ to hold, there should be a time instant in which *both* p and q hold. By contrast the present paper allows all four combinaisons of timing intervals, i.e. of the form $[a, b]$, $[a, b)$, $(a, b]$ and (a, b), but obtains a simplified translation by rewriting modalities timed by (a, b) into other modalities timed by intervals of the form $(0, c)$ only.

Although the simplification achieved in [54] by restricting the syntax and semantics to closed intervals will be appreciated by anyone comparing it with the present paper, we observe that from a practical perspective, left-closed right-open signals are not always sufficient [74]. Consider for example the *rising edge* formula $\uparrow p$ that should hold at moments where p changes its value to

[2]See also [32] for a corrected version of the *release* operator and its rewriting using *until*.

true. Such *events*, that is, formulae that are valid only at isolated time points, are very natural yet cannot be expressed without allowing signals with discontinuities of arbitrary form (left-, right-, or left- and right-). Reintroducing the original interpretations of *until* and *since* enables the natural encoding of events in the signal semantics under consideration.

The rest of the paper is organized as follows. In Section 2 we give an introduction to the idea of temporal testers and then show in Section 3 how they are used to translate LTL to automata. In Section 4 we introduce the semantic domain of signals, present the logic MITL and establish some valid rewriting rules on MITL formulae that allow us to later focus on the construction of testers for simple operators. The main result, the compositional translation is described in Section 5 followed by a summary and a short discussion of potential applications in Section 6.

2 TEMPORAL TESTERS

The standard methodology for checking whether all the behaviors of a finite-state system S modeled by an automaton $\mathcal{A}_S$ satisfy a specification expressed as a temporal property φ involves building a Büchi automaton $\mathcal{A}_{\neg\varphi}$ that *accepts* exactly all the (infinite) words that violate the property φ. The model checking problem, that is, deciding the language inclusion $L(\mathcal{A}_S) \subseteq L(\varphi)$ between the possible behaviors of $\mathcal{A}_S$ and the behaviors satisfying φ, reduces to checking whether the product automaton $\mathcal{A}_S \times \mathcal{A}_{\neg\varphi}$ accepts the empty language, implying that there exists no computation of S which violates φ.

In the discrete-time domain, the construction of $\mathcal{A}_{\neg\varphi}$ typically follows a tableau-based procedure based on *expansion formulae* that separate the variable values that have to hold at the *current* position from the future obligations which are propagated to the *next* position, for example $\Box\,\varphi \Leftrightarrow \varphi \wedge \bigcirc \Box\,\varphi$. The expansion rules rely heavily on the *next* operator $\bigcirc$ which allows to separate clearly current obligations from future ones. Obviously this idea cannot be directly applied to behaviors defined over a *dense* time domain.

The growing complexity of digital systems calls for more modular and compositional reasoning about them. We note that existing hierarchical and incremental design practices already provide opportunity for some non-negligible amount of specification reuse. Traditional tableau-based acceptors are not modular in nature. The lack of modularity is due to the fact that an acceptor $\mathcal{A}_\varphi$ provides information concerning the satisfaction of φ by the *entire* input sequence starting at position 0, but no information concerning satisfaction of φ by the input sequence suffixes starting at any position $t > 0$. In particular, when $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$ are the acceptors for formulae φ and ψ, respectively, there is no simple recipe to compose them to obtain an acceptor for the formula $\varphi\,\mathcal{U}\,\psi$. The property $\varphi\,\mathcal{U}\,\psi$ is satisfied iff there is a *future* position $t > 0$ where ψ is true, and that φ holds continuously at all positions t' such that $0 < t' < t$. The acceptors $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$ do not provide this information.

An alternative style of construction (see [75]) uses *alternating automata* [21, 23], automata that employ both existential and universal nondeterminism. The construction of alternating automata from formulae is, in some sense, more modular and elegant, however it is not compositional in the sense of this paper: the automaton for a formula may make *transitions* to the automata of its sub-formulae but it does not observe the evolution of their satisfiability over time. Since model-checkers deal only with existential nondeterminism, the universal nondeterminism has to be removed by a kind of subset construction [63] (also called *dealternation*) at exponential cost.

Our construction is based on *temporal testers*, an orthogonal solution to the problem of compositionality where additional structure imposes the responsibility of being composable on the automata for the sub-formulae [62, 69]. Consider a formula φ defined over propositional variables $p_1, \ldots, p_n$. A temporal tester $\mathcal{T}_\varphi$ for φ is a *transducer* whose input alphabet is $\mathbb{B}^n$ (vectors of n Boolean values 0 or 1), representing valuations of the propositional variables appearing in φ, and

whose output alphabet is $\mathbb{B} = \{0, 1\}$. While observing an input sequence w, the tester outputs a Boolean sequence u such that $u[t] = 1$ iff φ is satisfied at t, that is $(w, t) \models \varphi$. Hence, unlike an acceptor $\mathcal{A}_\varphi$ which tells us whether the entire input sequence satisfies φ, the temporal tester $\mathcal{T}_\varphi$ does so for *every suffix* of w. This stronger condition allows testers to compose naturally: we can view the output of $\mathcal{T}_\varphi$ as a propositional variable q satisfying $\square(q \leftrightarrow \varphi)$. For a formula φ which has φ_1 and φ_2 as sub-formulae we can then build a tester $\mathcal{T}_\varphi$ over input variables q_1 and q_2, which will take the outputs of $\mathcal{T}_{\varphi_1}$ and $\mathcal{T}_{\varphi_2}$ as inputs for $\mathcal{T}_\varphi$. The decomposition can be described in logic as replacing formula φ by the equisatisfiable formula $\varphi' \wedge \square(q_1 \leftrightarrow \varphi_1) \wedge \square(q_2 \leftrightarrow \varphi_2)$, where φ' is obtained from φ by replacing φ_1 by q_1 and φ_2 by q_2.[3] After repeating the process for all other non-atomic subformulae we obtain a formula of the form $q_n \wedge \bigwedge_{i=1}^{n} \square(q_i \leftrightarrow \varphi_i')$ where every φ_i' has only atomic sub-formulae. Taking the composition of testers for all φ_i (and projecting away the q_i variables) gives us a tester $\mathcal{T}_\varphi$ for φ. Further composing $\mathcal{T}_\varphi$ with an acceptor of proposition q_n yields an acceptor for φ. A construction of a network of testers for the formula $(p \wedge \bigcirc q)\,\mathcal{U}\,\square r$ is illustrated in Figure 1; for simplicity we label output signals by formulae they stand for in the place of fresh variable names.

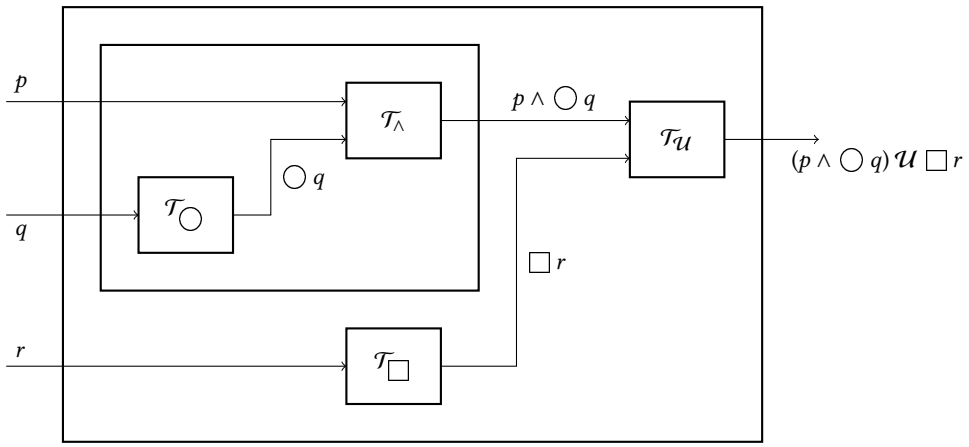

Fig. 1. Composition of temporal testers for $(p \wedge \bigcirc q)\,\mathcal{U}\,\square r$.

Below, we list some properties of temporal testers that make them particularly useful:

- The construction of temporal testers is completely modular. It suffices to build testers for basic temporal and logical operators, $\neg$, $\wedge$, $\bigcirc$ and $\mathcal{U}$ in the case of LTL. Testers for arbitrary formulae are constructed by synchronous, input-output composition of these building blocks.
- Temporal testers naturally support extensions of the specification language. Once a new language construct is introduced, its corresponding tester can be naturally composed with testers for existing operators. This feature has already been used to extend compositional construction of testers for LTL [45] with the regular expression-like operators of PSL [68] and with branching-time operators of CTL* [44]. Likewise the combination of future and past operators comes for free.
- Testers for specific properties that have been expressed directly by an automaton or a program without a formal logical description, or that have been optimized [24] can be combined with testers developed in a different way, as long as they produce the right output.

[3] This is reminiscent of Ceitin's procedure for transforming Boolean formulae to CNF

- Unlike certain tableau-based techniques, the construction of temporal testers does not require the existence of expansion formulae. This is particularly important for testers defined algorithmically and for real-time logics such as MITL where the meaningfulness of the *next* operator $\bigcirc$ is not evident.
- Although temporal testers are transducers that incorporate additional structure with respect to acceptors, the complexity of constructing such a tester for an arbitrary LTL formula is not worse than that of an acceptor. In its symbolic representation, the size of a tester is linear in the size of the formula. This implies that the worst-case state complexity is exponential for LTL formulae, which is an established lower bound.

The idea of transducers that output the truth value of a temporal formula at each position was first considered in [61, 62] as a conceptually simpler translation from temporal logics to ω-automata. The term *testers* was introduced in [45] who define such transducers for the purpose of efficient LTL-based formal verification. A similar idea was also considered in [22] in the context of symbolic implementation of a tableau construction. The observation that a Boolean variable such as the one output by a tester can replace the sub-formula itself in the context of model checking has been considered in [25]. Surprisingly, these techniques went unnoticed in the verification community until more recently. In [44] testers are extended to branching time, leading to a new CTL* model checking algorithm. The properties of temporal testers have been studied in detail with respect to acceptors and alternating automata in [69] and much of the material in this section is borrowed from it.

3 LINEAR TEMPORAL LOGIC

In this section we recall the construction of testers for the basic LTL operators (including *past* ones). We assume familiarity with LTL, automata over ω-words, and transducers. In brief, LTL is defined over a set of propositional variables $P = \{p_1, p_2, \ldots, p_k\}$ with Boolean operators $\neg$, $\wedge$ and temporal operators *next* $\bigcirc$, *previously* $\ominus$, *until* $\mathcal{U}$, and *since* $\mathcal{S}$. The semantics of LTL is over sequences $\mathbb{N} \mapsto 2^k$, through a satisfaction relation $\models$ between a pair (w, t) of a sequence w and time $t \in \mathbb{N}$, and a formula φ. This relation is defined inductively by letting

$$(w,t) \models \bigcirc \varphi \qquad \text{iff} \qquad (w,t+1) \models \varphi$$

$$(w,t) \models \varphi_1 \,\mathcal{U}\, \varphi_2 \qquad \text{iff} \qquad \exists t' \geq t,\ (w,t') \models \varphi_2 \text{ and } \forall t'' \in [t,t'),\ (w,t'') \models \varphi_1$$

and symmetrically for past operators $\ominus$ and $\mathcal{S}$.

We observe that the satisfaction of an LTL formula φ over propositions $p_1, \ldots, p_k$ by a sequence w at position t is a φ-dependent function of the truth values of $p_1, \ldots, p_k$ at some other positions. The satisfaction relation can be viewed as the *characteristic function* χ^φ which maps sequences $w : \mathbb{N} \to \mathbb{B}^k$ into sequences $u = \chi^\varphi(w) : \mathbb{N} \to \mathbb{B}$ such that for every $t \geq 0$, $u[t] = 1$ iff $(w, t) \models \varphi$.[4] For formulae φ of the form $f(p)$ or $f(p_1, p_2)$ for some temporal or propositional operator f, by slight abuse of notation we simply write χ^f in place of χ^φ. The inductive semantics of LTL can then be seen as a recipe for building the characteristic function of φ from the characteristic functions of its sub-formulae, as illustrated in Figure 1. These characteristic functions which are to be realized by the temporal testers are instances of the class of *sequential functions* (transducers), functions that map sequences to sequences. A particular sub-class of sequential functions are the *causal* (sometime called *retrospective* [74]) functions.

Definition 3.1 (Causal Sequential Functions). A sequential function $f : A^\omega \to B^\omega$ is said to be *causal* if for every $u \in A^*$ and $v, v' \in B^*$ such that $|u| = |v| = |v'|$ and every $x, x' \in A^\omega$ and

[4] Assuming the Boolean value in u is accessed by propositional variable q, recall that the relation between u and w can also be expressed by the formula $\square(q \leftrightarrow \varphi)$.

$y, y' \in B^\omega$:

$$f(u \cdot x) = v \cdot y \text{ and } f(u \cdot x') = v' \cdot y' \text{ implies } v = v'.$$

In other words, the value of $f(w)$ at time t may depend only on the values $\{w[t'] : t' \leq t\}$. Causal functions are realized naturally by *deterministic* automata with output (sequential synchronous transducers) that produce the next output symbol as they read the next input symbol. The semantics of the past fragment of LTL can be expressed using causal functions, because the satisfaction of both *previously* $\ominus$ and *since* $\mathcal{S}$ operators *now* (at position t) is determined according to what has happened *until now* (positions $t' \leq t$).

The characteristic function of $\ominus$, is nothing but a *shift* operator, known in other contexts as the unit delay operator z^{-1}, defined as $u[t+1] = w[t]$ for $t > 0$, and $u[0] = 0$. The temporal tester for $\ominus p$, shown in Figure 2-(a), is a simple one-bit input-driven shift register. At each time instant this transducer reads the current value of p, memorizes it by taking a transition to the appropriate target state, and outputs the previous value as encoded by the source state of the transition. By convention, we use q as the output variable. Being at state s_0 means that p held in the previous step, while being at s_1 means that it did not hold. When the new value of the input is p, the automaton will move to state s_1, while if it is $\neg p$ it remains in s_0. The tester is input-deterministic, as from any state there is a single outgoing transition for a given input symbol.

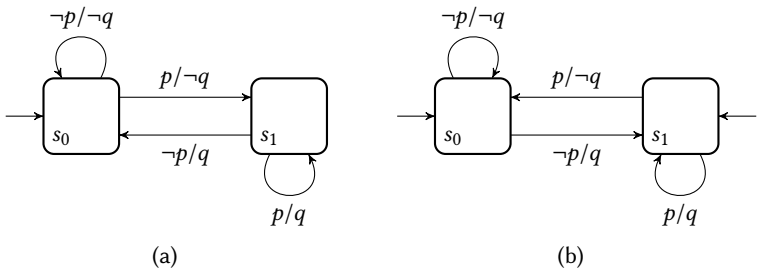

Fig. 2. Temporal testers for LTL: (a) operator $\ominus$; (b) operator $\bigcirc$.

On the other hand, the characteristic functions associated with future LTL operators are not causal as the satisfaction at t may depend on satisfaction at some $t' > t$. The output of the *next* operator $\bigcirc$ at time t depends on the input at $t+1$ and, even worse, the output of the *until* operator $\mathcal{U}$ at t may depend on input values at arbitrary larger t'.

One can think of two ways to realize acausal sequential functions. The first approach, which works for operators with a *bounded* level of acausality, for example $\bigcirc^d$ (the *next* operator nested d times), is to dissociate the time scales of the input and the output, that is, let the automaton ignore the first d input symbols, and then let $u[t] = w[t+d]$. Unfortunately, this approach does not work for unbounded acausality and also does not compose well. In the alternative approach that we use, the temporal testers respond to the input *synchronously*, but since at time t the information might not be sufficient to determine the output, the tester has to "guess" the output nondeterministically and split the computation into two runs, one that *predicts* $u[t] = 1$ and one that predicts $u[t] = 0$. Each of the runs needs to remember the predictions it has made so far and, progressively, abort runs whose predictions turn out do be false. An automaton for an operator with acausality of depth d may need to memorize up to 2^d past predictions.

The similarity between remembering past observations (in a shift register) and remembering predictions is no coincidence. The temporal tester for $\bigcirc$, depicted in Figure 2-(b) can be obtained by *reversing* the transitions of the automaton for $\ominus$. The automaton for $\bigcirc p$ is output-deterministic

and its state memorizes the prediction it made in the previous step. The prediction is used to abort, in the next step, runs whose predictions turned out to be wrong.

To understand how such an acausal tester works, let us look at Figure 3-(a) which shows the $\bigcirc$-tester in an extended form where *abortions* due to wrong predictions are made explicit. States s_1 indicates that the prediction made in the current step is q, hence from this state, observing $\neg p$ contradicts the prediction and the run is aborted (*abort* transition). Input p confirms the prediction and the automaton splits the remaining run into two by moving nondeterministically to s_0 and s_1, thus generating two predictions for the next value and so on. For every ω-sequence w, only *one* infinite run survives and its output is $u = \chi^{\bigcirc}(w)$. An initial prefix of a sample run is shown in Figure 3-(b).

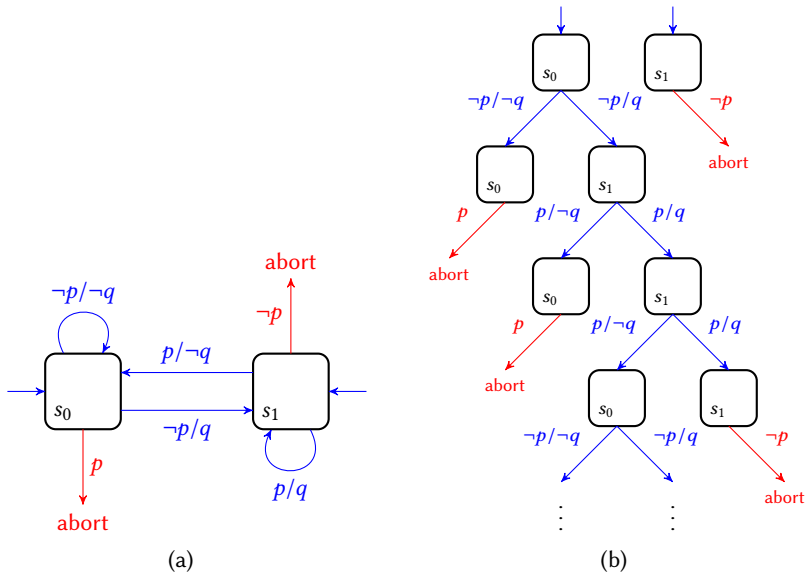

Fig. 3. Behavior of temporal testers: (a) the tester for $\bigcirc$ with input p and output q; (b) An initial fragment of the behavior of this tester for an input sequence 0110 . . . producing the output 110 . . .

The output of the past temporal tester for $p_1 \mathcal{S} p_2$ is again fully determined by the observed past history. At positions where $\neg p_1 \wedge \neg p_2$ is observed, the property does not hold and the tester outputs $\neg q$. Likewise, when p_2 is observed, the output is trivially determined to be q. In a state where $p_1 \wedge \neg p_2$ is observed, the formula $p_1 \mathcal{S} p_2$ can be either satisfied or falsified, depending on the previous observations, that is whether p_1 has been continuously holding from the last time p_2 was true. This situation is reflected by two states s_0 and s_1. In s_0, no state p_2 has yet been observed in the period starting immediately before p_1 started holding, and thus $\neg q$. In s_1, state p_2 has been observed and p_1 since, hence q.

The situation with $p_1 \mathcal{U} p_2$, although symmetric to $p_1 \mathcal{S} p_2$, is more involved because a priori, due to the unbounded future horizon, one might need to generate and memorize 2^ω predictions. However, the semantics of *until* implies that at most *two* confirmable predictions may co-exist simultaneously.

Lemma 3.2. *Let $u = \chi^{p_1 \mathcal{U} p_2}(w)$. Then for every t such that $(w, t) \models p_1 \wedge \neg p_2$, $u[t] = u[t+1]$.*

Proof. There are three possibilities: (1) The earliest $t' > t+1$ such that $(w, t') \models \neg p_1 \vee p_2$ satisfies $(w, t') \models p_2$. In that case, the property is satisfied both at t and $t+1$; (2) The same t'

satisfies $(w, t') \models \neg p_1 \wedge \neg p_2$ and the property is violated both at t and $t+1$; (3) $(w, t') \models p_1 \wedge \neg p_2$ for every $t' > t+1$ and the property is falsified from both time points. □

This fact is reflected by the tester of Figure 4-(b). The tester proceeds by guessing at every t whether $p_1 \,\mathcal{U}\, p_2$ will hold at time $t+1$ by moving to s_1 (*will hold*) or to s_0 (*will not hold*). Simultaneously it outputs the truth value of $p_1 \,\mathcal{U}\, p_2$ at time t in variable q when previous predictions are not contradicted by the input, and aborts otherwise. At time instants where $\neg p_1 \wedge \neg p_2$ is observed, the value of the output is determined to be $\neg q$. In that case from state s_1 the tester aborts because the prediction is contradicted, and from state s_0 it makes a new prediction by moving to s_1 or s_0. Likewise, at time instants when p_2 is observed the output is determined to be q, and the tester either aborts (from s_0) or makes a new prediction (from s_1). In the situation where $p_1 \wedge \neg p_2$ holds, nothing can be concluded from the input as to whether $p_1 \,\mathcal{U}\, p_2$ holds, and the situation is handled according to Lemma 3.2. The output is dictated by the previous prediction, that is, the tester outputs q from state s_1 and $\neg q$ from state s_2. To check a positive prediction, we must observe p_2 from state s_0, and to check a negative prediction we must either observe $\neg p_1 \wedge \neg p_2$ from state s_0, or $p_1 \wedge \neg p_2$ holding forever in state s_0. To prevent the tester from being in state s_1 when $p_1 \wedge \neg p_2$ repeats forever we use an edge Büchi condition [33], that includes all edges of the tester except for the self-loop labeled p_1 in state s_1 (see Figure 4-(b)). This ensures that this particular edge is consecutively taken at most a finite number of times.

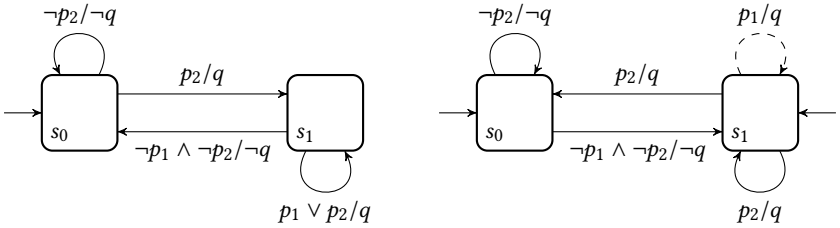

Fig. 4. Temporal testers for LTL: (a) operator $\mathcal{S}$; (b) operator $\mathcal{U}$. The acceptance condition is placed on edges: a dashed edge is defined as *unstable* and can only be taken finitely many consecutive times [62]; this is equivalent to the Büchi condition that at least one plain edge must be taken infinitely many times. Note that acceptance here has nothing to do with the satisfaction of the property but whether the sequential function $u = \chi^{\mathcal{U}}(w)$ computed by the run is correct.

Testers for complex formulae are constructed by standard input/output composition as shown in Section 2. With these four testers, and the trivial testers for the Boolean operators, one can indeed build testers for arbitrary (past and future) LTL formulae.

4 SIGNALS AND THEIR TEMPORAL LOGIC

Extending the construction of temporal testers from discrete to dense time requires significant adaptations of the semantic domain, the logic and the automata. The interaction between discrete events and dense time may give rise to certain well-known anomalies that should be carefully avoided.

Let the time domain $\mathbb{T}$ be the set of nonnegative real numbers. The Minkowski sum $I \oplus J$ over $\mathbb{T}$ of two sets I, J is the set $\{r + s \in \mathbb{T} \mid r \in I, s \in J\}$. In the special case of closed intervals one has $[a, b] \oplus [c, d] = [a + c, b + d]$, and in the general case since $x < a$ and $y \leq b$ imply $x + y < a + b$, the Minkowski sum behaves according to Table 1. We also use the notation

$I \ominus J = \{r - s \in \mathbb{T} \mid r \in I, s \in J\}$ to denote the Minkowski difference within $\mathbb{T}$. The notations $t \oplus I$ and $t \ominus I$ are shorthands for $\{t\} \oplus I$ and $\{t\} \ominus I$, respectively. For any bounded interval I we will also write $|I|$ for $\sup I - \inf I$.

4.1 Signals

A (multi-dimensional) Boolean *signal* is a function $w : \mathbb{T} \to A$. In this paper we focus on the case where the alphabet A is a set $\mathbb{B}^n$ of Boolean vectors over n variables. A *bounded signal* is a function $w : T \to A$ whose domain of definition is $T \subseteq \mathbb{T}$ is such that $\inf T = 0$ and $\sup T < +\infty$. The length of a signal w is denoted $|w| = \sup T$. For a bounded domain T and a domains T' such that T right-closed and T' left-open or vice-versa, the concatenation of signals $w : T \to A$ and $w' : T' \to A$ is defined as the signal $w \cdot w' : T \oplus T' \to A$ such that $(w \cdot w')[t] = w[t]$ if $t \in T$, $w'[t - \sup T]$ otherwise.

Each signal can be decomposed into several *point* segments defined at time 0 and *open* segments defined over intervals of the form $(0, r)$ for some duration $r > 0$. A point-segment partition of $\mathbb{T}$ is an alternating sequence of adjacent points and open intervals of the form

$$J = \{t_0\}, (t_0, t_1), \{t_1\}, (t_1, t_2), \ldots$$

with $t_0 = 0$ and $t_i < t_{i+1}$. A signal is *well-behaving* if it admits a compatible time partition. We exclude so-called *Zeno* signals, and in general those that have infinitely many discontinuities over a bounded time interval. With respect to such a given time partition, a well-behaving signal w can be written as an alternating concatenation of point segments and open segments:

$$w = \dot{\sigma}_0 \cdot \sigma_0 \cdot \dot{\sigma}_1 \cdot \sigma_1 \cdots$$

where $\dot{\sigma}_i$ is the point segment at t_i and σ_i is the open segment between times t_i and t_{i+1}. A time partition is *compatible* with a signal w if the value of w is uniform in each open interval (t_i, t_{i+1}).

Given a value $a \in A$ and a duration $r \in \mathbb{T}$ we denote by a^r the open signal segment defined over $T = (0, r)$ with constant value a. By slight abuse of notation, we also write a to denote the point signal segment $\{0\} \to A$ with value $a \in A$. The decomposition of a well-behaving signal w relative to a compatible time partition can be written

$$w = \dot{w}_0 \cdot w_0^{r_0} \cdot \dot{w}_1 \cdot w_1^{r_1} \cdots$$

where each $\dot{w}_i \in A$ denotes the value of w at point t_i and each $w_i \in A$ denotes the value of the signal in the interval (t_i, t_{i+1}) of duration $r_i = t_{i+1} - t_i$. The coarsest time partition compatible with a well-behaved signal w is obtained from the discontinuities of w, and denoted J_w. When the signal decomposition is relative to J_w, then for all $i > 0$ by definition we have $\sigma_{i-1} \neq \dot{\sigma}_i$ or $\dot{\sigma}_i \neq \sigma_i$.

Boolean signal components can be combined and separated using the standard pairing and projection operators. Let $w_p : \mathbb{T} \to \mathbb{B}$, $w_q : \mathbb{T} \to \mathbb{B}$ and $w_{pq} : \mathbb{T} \to \mathbb{B}^2$ be signals. The pairing function is defined as

$$w_p \parallel w_q = w_{pq} \quad \text{if} \quad \forall t \in \mathbb{T},\ w_{pq}[t] = (w_p[t], w_q[t])$$

and its inverse operation, projection as:

$$w_p = w_{pq}|_p \qquad\qquad w_q = w_{pq}|_q$$

Table 1. Bounds in the Minkowski sum of two intervals.

$\oplus$	$[c$	$(c$
$[a$	$[a+c$	$(a+c$
$(a$	$(a+c$	$(a+c$

$\oplus$	$d)$	$d]$
$b)$	$b+d)$	$b+d)$
$b]$	$b+d)$	$b+d]$

Signal transducers are functions that map signals to signals. They can be memoryless such as the pointwise extensions of Boolean operations or more general ones realized by (timed) automata. The definition of causal signal transducers is similar to the definition for sequence transducers, see Definition 3.1.

Note that the number of point segments in w_{pq} is at most the *sum* of the number of point segments in w_p and w_q, and that the number of point segments in $f(w_p, w_q)$, for the pointwise extension of a Boolean operator f is at most that of w_{pq}. Hence well-behaving signals are closed under pairing, projection and Boolean operations.

4.2 MITL: Real-time Temporal Logic

4.2.1 Syntax and Semantics. We consider the MITL logic with both *future* and *past* operators. The syntax of MITL is defined by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2 \mid \varphi_1 \,\mathcal{S}_I\, \varphi_2$$

where p belongs to a set P of propositions and I is an (unbounded or bounded) interval of $\mathbb{T}$ with rational end-points.[5] As in LTL the basic MITL operators can be used to derive other standard Boolean and temporal operators, in particular the time-constrained *eventually*, *always*, *once*, and *historically* operators:

$$\Diamond_I \varphi \equiv \top \,\mathcal{U}_I\, \varphi \qquad \Box_I \varphi \equiv \neg \Diamond_I \neg\varphi$$
$$\Diamond\!\!\!-_I \varphi \equiv \top \,\mathcal{S}_I\, \varphi \qquad \Box\!\!\!-_I \varphi \equiv \neg \Diamond\!\!\!-_I \neg\varphi$$

where $\top$ is the constant *true*.

The semantics of an MITL formula φ with respect to a multi-dimensional Boolean signal w is described via the satisfiability relation $(w,t) \models \varphi$, indicating that the signal w satisfies φ at time t, according to the following recursive definition.

$$\begin{array}{lll}
(w,t) \models p & \text{iff} & t \in w_p[t] = 1 \\
(w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \,\mathcal{U}_I\, \varphi_2 & \text{iff} & \exists t' \in t \oplus I,\ (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t,t'),\ (w,t'') \models \varphi_1 \\
(w,t) \models \varphi_1 \,\mathcal{S}_I\, \varphi_2 & \text{iff} & \exists t' \in t \ominus I,\ (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t',t),\ (w,t'') \models \varphi_1
\end{array}$$

A formula φ is satisfied by w if $(w,0) \models \varphi$. The definitions of $\mathcal{U}_I$ and $\mathcal{S}_I$ are *strict* as originally proposed in [5], meaning that the moment t' when φ_2 is true is required to be strictly after t, and that φ_1 need not hold at t. Untimed *strict* temporal operators $\mathcal{U}$ and $\mathcal{S}$ can be expressed using the timed operators where the interval is $(0,\infty)$. We can define *non-strict* untimed temporal operators $\underline{\mathcal{U}}$ and $\underline{\mathcal{S}}$ (which are the commonly-used interpretations of $\mathcal{U}$ and $\mathcal{S}$ in LTL) in terms of the strict ones as follows:

$$\varphi_1 \,\mathcal{U}\, \varphi_2 \equiv \varphi_1 \,\mathcal{U}_{(0,\infty)}\, \varphi_2 \qquad \varphi_1 \,\mathcal{S}\, \varphi_2 \equiv \varphi_1 \,\mathcal{S}_{(0,\infty)}\, \varphi_2$$
$$\varphi_1 \,\underline{\mathcal{U}}\, \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)) \qquad \varphi_1 \,\underline{\mathcal{S}}\, \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge (\varphi_1 \,\mathcal{S}\, \varphi_2)).$$

Note that $\underline{\mathcal{U}}$ differs from $\mathcal{U}_{[0,\infty)}$, which is equivalent to $\varphi_2 \vee \varphi_1 \,\mathcal{U}\, \varphi_2$.

Let us remark that the original logic MTL [48] for which MITL is a restriction also allows "punctual" intervals of the form $[a,a]$ in temporal modalities. To see why this is problematic in dense time consider the operator $\Diamond\!\!\!-_{[a,a]}$. This operator, viewed as a signal transducer is a *shift*: its output at t is the value of its input at time $t-a$. To realize this operator we would need a device

[5] As a general remark concerning timed automata and logics, by suitable scaling every MITL formula and finite timed automaton can be converted to one where a and b are integers.

which can "memorize" the value of the input signal in a time window of length a. Without further assumptions on the signal, such a memorization is beyond the capabilities of any automaton with a finite number of states and clocks. The same applies to the future operator $\Diamond_{[a,a]}$, which must "predict" the value of the input signal instead of memorizing it. However that if one knows in advance a bound on the *variability* of the input, this operator can be realized by a finite timed automaton. In particular, for formulae φ whose shortest *true* time segments always last more than a, the formula $\Diamond_{[a,a]}\,\varphi$ can be replaced by $\Diamond_{[0,a]}\,\Box_{[0,a]}\,\varphi$. We will use similar facts in the following.

4.2.2 Expressing Events. As defined, MITL does not provide constructs that allow to reason explicitly about *instantaneous events*, which can be viewed as taking place in singular intervals of zero duration. A natural way to introduce them is to consider the instants when a signal changes its value. The (*strict*) *until* and *since* operators allow us to define *next* and *previously* operators that are suitable in our dense-time setting[6] as follows:

$$\bigcirc\,\varphi \equiv \varphi\,\mathcal{U}\,\varphi \qquad\qquad \ominus\,\varphi \equiv \varphi\,\mathcal{S}\,\varphi.$$

Intuitively $\bigcirc\,\varphi$ (resp. $\ominus\,\varphi$) means that φ holds immediately after now (resp. held just before now). We then propose two unary operators, *rise* $\uparrow$ and *fall* $\downarrow$ that hold at the rising and falling edges of a Boolean signal, respectively. Given that we allow singular points to be equal to their left neighborhood, $\uparrow p$ may hold at t even if $p[t] = 0$ as illustrated in Figure 5. More precisely, $\uparrow\varphi$ holds at t if φ is true in a right neighborhood of t and false in a left neighborhood of t. A similar definition is taken for falling edges. These operators can be expressed in MITL as follows:

$$\uparrow\varphi \equiv (\varphi \wedge \ominus\,\neg\varphi) \vee (\neg\varphi \wedge \bigcirc\,\varphi) \qquad\qquad \downarrow\varphi \equiv (\neg\varphi \wedge \ominus\,\varphi) \vee (\varphi \wedge \bigcirc\,\neg\varphi).$$

Note that this requires both strict-future and strict-past temporal operators.

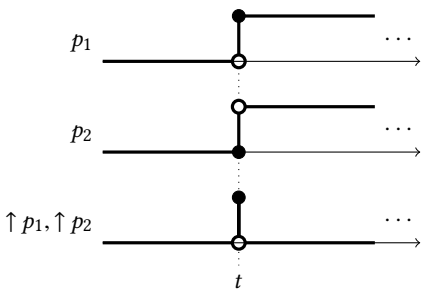

Fig. 5. Two signals p_1 and p_2 that differ at time t where both $\uparrow p_1$ and $\uparrow p_2$ hold.

4.2.3 Rewrite Rules. In what follows we show that we need not build testers for all 12 temporal operator variants ($\mathcal{U}_I$ and $\mathcal{S}_I$, each with all types of intervals) and we can restrict ourselves to 4 simpler ones. We start with the following lemma, also proved in [30, 38], which shows that the timed *until* can be expressed by a combination of untimed *until* and bounded *eventually*.

For any formulae φ and ψ, we write $\varphi \Leftrightarrow \psi$ when for all signals w and times $t \in \mathbb{T}$, $(w,t) \models \varphi$ iff $(w,t) \models \psi$.

[6]These definitions are also compatible with the standard semantics of *next* and *previously* in a discrete time setting when using strict *until* and *since* in that context.

Lemma 4.1 ($\mathcal{U}_I$ as combination of $\mathcal{U}$ and $\Diamond_J$ with $\sup J < \infty$). *For any rational constants a, b, c such that $0 \le a < b < \infty$ and $0 < c < \infty$, we have*

$$\varphi_1 \,\mathcal{U}_{(a,b)}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{U}_{(a,\infty)}\, \varphi_2 \wedge \Diamond_{(a,b)}\, \varphi_2 \qquad \varphi_1 \,\mathcal{U}_{[a,b)}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{U}_{[a,\infty)}\, \varphi_2 \wedge \Diamond_{[a,b)}\, \varphi_2$$
$$\varphi_1 \,\mathcal{U}_{(a,b]}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{U}_{(a,\infty)}\, \varphi_2 \wedge \Diamond_{(a,b]}\, \varphi_2 \qquad \varphi_1 \,\mathcal{U}_{[a,b]}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{U}_{[a,\infty)}\, \varphi_2 \wedge \Diamond_{[a,b]}\, \varphi_2$$
$$\varphi_1 \,\mathcal{U}_{(c,\infty)}\, \varphi_2 \Leftrightarrow \Box_{(0,c]}(\varphi_1 \wedge \varphi_1 \,\mathcal{U}\, \varphi_2) \qquad \varphi_1 \,\mathcal{U}_{[c,\infty)}\, \varphi_2 \Leftrightarrow \Box_{(0,c)}\, \varphi_1 \wedge \Box_{(0,c]}(\varphi_1 \,\underline{\mathcal{U}}\, \varphi_2).$$

Proof. We prove the first and fifth equivalences, others are similar.

One direction of the first equivalence follows directly from the semantics of timed *until*, so we consider only the other direction. When formula $\Diamond_{(a,b)}\, \varphi_2$ holds at time t in combination with $\varphi_1 \,\mathcal{U}_{(a,\infty)}\, \varphi_2$, then φ_1 holds until some time t_1 after $t + a$ where φ_2 holds, and in addition there is $t_2 \in (t + a, t + b)$ where φ holds. If $t_1 \in (t + a, t + b)$ then t_1 is a witness of $\varphi_1 \,\mathcal{U}\, \varphi_2$, otherwise $t_1 \ge t + b$ and thus φ holds until $t_2 < t_1$ so that t_2 is a witness of $\varphi_1 \,\mathcal{U}\, \varphi_2$, and we are done.

Similarly, one direction of the fifth equivalence follows from the semantics $\mathcal{U}$ and $\Box$. In the other direction, assume $\Box_{(0,c]}(\varphi_1 \wedge \varphi_1 \,\mathcal{U}\, \varphi_2)$ holds at t. Then φ_1 holds throughout $(t, t + c]$ and also holds until some $t_2 \in (t + c, \infty)$ where φ_2 holds, because in particular $\varphi_1 \,\mathcal{U}\, \varphi_2$ holds at $t + c$. Thus φ_1 holds over (t, t_2) for some t_2 where φ_2 holds, which means that $\varphi_1 \,\mathcal{U}\, \varphi_2$ holds at t. □

For past operators we have:

Lemma 4.2 ($\mathcal{S}_I$ as combination of $\mathcal{S}$ and $\Diamond\!\!\!\!-_J$ with $\sup J < \infty$). *For any rational constants a, b, c such that $0 \le a < b < \infty$ and $0 < c < \infty$, we have*

$$\varphi_1 \,\mathcal{S}_{(a,b)}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{S}_{(a,\infty)}\, \varphi_2 \wedge \Diamond\!\!\!\!-_{(a,b)}\, \varphi_2 \qquad \varphi_1 \,\mathcal{S}_{[a,b)}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{S}_{[a,\infty)}\, \varphi_2 \wedge \Diamond\!\!\!\!-_{[a,b)}\, \varphi_2$$
$$\varphi_1 \,\mathcal{S}_{(a,b]}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{S}_{(a,\infty)}\, \varphi_2 \wedge \Diamond\!\!\!\!-_{(a,b]}\, \varphi_2 \qquad \varphi_1 \,\mathcal{S}_{[a,b]}\, \varphi_2 \Leftrightarrow \varphi_1 \,\mathcal{S}_{[a,\infty)}\, \varphi_2 \wedge \Diamond\!\!\!\!-_{[a,b]}\, \varphi_2$$
$$\varphi_1 \,\mathcal{S}_{(c,\infty)}\, \varphi_2 \Leftrightarrow \boxminus_{(0,c]}(\varphi_1 \wedge \varphi_1 \,\mathcal{S}\, \varphi_2) \qquad \varphi_1 \,\mathcal{S}_{[c,\infty)}\, \varphi_2 \Leftrightarrow \boxminus_{(0,c)}\, \varphi_1 \wedge \boxminus_{(0,c]}(\varphi_1 \,\underline{\mathcal{S}}\, \varphi_2).$$

Proof. Symmetrical with Lemma 4.1. □

Consequently, the operators $\mathcal{U}$, $\mathcal{S}$, $\Diamond_I$ and $\Diamond\!\!\!\!-_I$, where I ranges over the interval types $[a, b]$, $[a, b)$, $(a, b]$ and (a, b), are sufficient to express any MITL property.

We can still reduce the number and complexity of tester types using the following equivalences first mentioned by [38] in the setting of *quantitative monadic logic* (QTL). In that work, the set of timed modalities is reduced to the mere $\Diamond_{(0,1)}$ and $\Diamond\!\!\!\!-_{(0,1)}$. This presumes that only integer timing constants are allowed. In the present paper, we assume rational constants.

Lemma 4.3 ($\Diamond_J$ as combination of $\Diamond_K$ with $\inf K = 0$). *For any rational constants a, b, c such that $0 < c \le b - a$, we have*

$$\Diamond_{(a+c,b+c)}\, \varphi \Leftrightarrow \Diamond_{(0,c)}\, \Box_{(0,c)}\, \Diamond_{(a,b)}\, \varphi \qquad \Diamond_{[a+c,b+c)}\, \varphi \Leftrightarrow \Diamond_{[0,c)}\, \Box_{(0,c]}\, \Diamond_{[a,b)}\, \varphi$$
$$\Diamond_{(a+c,b+c]}\, \varphi \Leftrightarrow \Diamond_{(0,c]}\, \Box_{[0,c)}\, \Diamond_{(a,b]}\, \varphi \qquad \Diamond_{[a+c,b+c]}\, \varphi \Leftrightarrow \Diamond_{[0,c]}\, \Box_{[0,c]}\, \Diamond_{[a,b]}\, \varphi.$$

Proof. We only show the first equivalence; other equivalences are similar.

(⇒) Assume that $\Diamond_{(a+c,b+c)}\, \varphi$ holds at time t, that is, φ holds at some time $t' \in (t + a + c, t + b + c)$. Then $\Diamond_{(a,b)}\, \varphi$ holds true over $(t' - b, t' - a)$, and thus $\Box_{(0,c)}\, \Diamond_{(a,b)}\, \varphi$ holds at time $t' - b \in (t, t + c)$ which means that $\Diamond_{(0,c)}\, \Box_{(0,c)}\, \Diamond_{(a,b)}\, \varphi$ holds at time t.

(⇐) We show the contrapositive. Assume that φ is false over $(t + a + c, t + b + c)$. Then $\Diamond_{(a,b)}$ is false at time $t + c$, and $\Box_{(0,c)}\, \Diamond_{(a,b)}\, \varphi$ is false over $(t, t + c)$, so that the right-hand side of the equivalence is false at time t.

□

For the past operators, we also have:

Lemma 4.4 ($\diamondsuit_J$ as combination of $\diamondsuit_K$ with $\inf K = 0$). *For any rational constants a, b, c such that $0 < c \leq b - a$, we have*

$$\diamondsuit_{(a+c,b+c)}\,\varphi \Leftrightarrow \diamondsuit_{(0,c)}\,\boxminus_{(0,c)}\,\diamondsuit_{(a,b)}\,\varphi \qquad \diamondsuit_{[a+c,b+c)}\,\varphi \Leftrightarrow \diamondsuit_{[0,c)}\,\boxminus_{(0,c]}\,\diamondsuit_{[a,b)}\,\varphi$$
$$\diamondsuit_{(a+c,b+c]}\,\varphi \Leftrightarrow \diamondsuit_{(0,c]}\,\boxminus_{[0,c)}\,\diamondsuit_{(a,b]}\,\varphi \qquad \diamondsuit_{[a+c,b+c]}\,\varphi \Leftrightarrow \diamondsuit_{[0,c]}\,\boxminus_{[0,c]}\,\diamondsuit_{[a,b]}\,\varphi.$$

Proof. Symmetrical with Lemma 4.3. □

Finally, closed and semi-closed intervals can be eliminated using the equivalence

$$\diamondsuit_{(0,a]}\,\varphi \Leftrightarrow \diamondsuit_{(0,a)}\,\varphi \vee (\bigcirc\,\diamondsuit_{(0,a)}\,\varphi \wedge \neg\varphi\,\mathcal{U}\,\varphi)$$

alongside $\diamondsuit_{[0,a)}\,\varphi \Leftrightarrow \varphi \vee \diamondsuit_{(0,a)}$ and $\diamondsuit_{[0,a]}\,\varphi \Leftrightarrow \varphi \vee \diamondsuit_{(0,a]}\,\varphi$. Here, the role of $\neg\varphi\,\mathcal{U}\,\varphi$ is to rule out the case where φ start holding immediately after a but does not hold at a. Symmetrical equivalences hold for past operators.

Putting all these results together, we obtain the main result of this section:

Proposition 4.5 (Basic MITL Operators). *Any MITL formula can be rewritten into an equivalent formula, which only uses temporal operators $\mathcal{U}$, $\mathcal{S}$, $\diamondsuit_{(0,a)}$, $\diamondsuit_{(0,a)}$ for rational constants $a > 0$.*

5 TIMED AUTOMATA

We use a variant of *timed automata* [3], which differ from their classical definition in several ways. Our automata are signal transducers that input and output multi-dimensional *dense-time* Boolean signals. The input and output valuations are associated with both automata locations and transitions. This allows a clean synchronization of the runs of the automaton (and their induced point-segment time partitions) with input and output signals.

Let $X = \{x_1, \ldots, x_n\}$ be a set of clock variables, each ranging over the nonnegative reals $\mathbb{T}$. A *configuration* of a timed automaton is a pair of the form (s, v) where s is the control state (or location) of the automaton and v the clock valuation. For a clock valuation $v = (v_1, \ldots, v_n) \in \mathbb{T}^n$, $v + t$ is the valuation $(v'_1, \ldots, v'_n)$ such that $v'_i = v_i + t$ for all $i = 1 \ldots n$. It represents the values of clocks after spending t time in a location starting from valuation v. Given $Z \subseteq X$, $v[Z]$ is the valuation $(v'_1, \ldots, v'_n)$ such that $v'_i = 0$ if $x_i \in Z$, else $v'_i = v_i$ for all $i = 1 \ldots n$. It represents the values of clocks after reseting those in Z to 0. A *clock constraint* is a Boolean combination of conditions of the form $x \leq c$ and $x \geq c$ for some rational constant c and clock variable $x \in X$.

Definition 5.1 (Timed Transducer). A *timed transducer* is a tuple $\langle S, \underline{s}, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$, where

- S is a finite set of discrete *locations* and $\underline{s} \notin S$ is the *initial* location;
- P and Q are finite sets of *input* and *output* variables;
- X is a finite set of *clock* variables;
- ι is a mapping from states in S to clock constraints over X also called *invariants*;
- Δ is the *transition* relation consisting of elements of the form $\delta = (s, g, Z, s')$, where
  - $s \in S \cup \{\underline{s}\}$ and $s' \in S$ are locations,
  - the *guard* g is a clock constraint over X,
  - the *reset instruction* Z is a subset of X;
- The *input labeling* λ is a function from $S \cup \Delta$ to Boolean combinations over P;
- The *output labeling* γ is a function from $S \cup \Delta$ to Boolean combinations over Q;
- $\mathcal{F} \subseteq 2^{S \cup \Delta}$ is a generalized Büchi acceptance condition.

Intuitively, a *run* of a timed transducer consists of an alternation of *discrete steps*, where a transition whose guard is satisfied is taken, and *time steps*, where the transducer stays in a location s for some duration, provided that the invariant $\iota(s)$ holds. We also need to establish a relation between a run of the transducer, an input signal w which induces it and an output signal u which is

generated during the run. First, we associate with every location and transition a Boolean constraint λ on the valuations of input variables. During a time step of duration r in a location s, the transducer reads an open segment σ of w of length r, whose values are required to satisfy $\lambda(s)$. While taking a transition δ, the transducer reads a point segment $\dot{\sigma}$ in the input signal w, whose value must satisfy $\lambda(\delta)$. Likewise, we associate with every location and transition a Boolean constraint γ over the output variables of the transducer: when in location s for duration r, the transducer writes an open segment ρ of length r to u according to $\gamma(s)$, and while taking a transition δ it writes $\dot{\rho}$ satisfying $\gamma(\delta)$ to u.

Formally, a step of a timed transducer is one of the following:

- A time step: $(s, v) \xrightarrow{\sigma/\rho} (s, v')$, for some duration $r = |\sigma| = |\rho|$ such that $v' = v + r$, $\sigma[t] \models \lambda(s)$, $\rho \models \gamma(s)$, and $v + t \models \iota(s)$ for all $t \in (0, r)$;
- A discrete step: $(s, v) \xrightarrow{\dot{\sigma}/\dot{\rho}} (s', v')$, for some transition $\delta = (s, g, Z, s') \in \Delta$ such that $v' = v[Z]$, $\dot{\sigma} \models \lambda(\delta)$, $\dot{\rho} \models \gamma(\delta)$, and $v \models g$.

A *run* of the transducer starting from the initial configuration (s, v) for some arbitrary v over an input signal w is a finite or infinite sequence of alternating discrete and time steps of the form

$$(\underline{s}, v) \xrightarrow{\dot{\sigma}_0/\dot{\rho}_0} (s_0, v_0) \xrightarrow{\sigma_0/\rho_0} (s_0, v_0 + r_0) \xrightarrow{\dot{\sigma}_1/\dot{\rho}_1} (s_1, v_1) \xrightarrow{\sigma_1/\rho_1} (s_1, v_1 + r_1) \dots$$

such that $w = \dot{\sigma}_0 \cdot \sigma_0 \cdot \dot{\sigma}_1 \cdot \sigma_1 \cdots$, inducing the output signal $u = \dot{\rho}_0 \cdot \rho_0 \cdot \dot{\rho}_1 \cdot \rho_1 \cdots$. It is accepting when it satisfies the generalized Büchi acceptance condition as follows: for all $F \in \mathcal{F}$, the set of absolute time instants at which the run visits a state or edge in F is unbounded. We say that $\mathcal{T}$ is *functional* when for every w there exists a unique u such that $\mathcal{T}$ has a run over w inducing u. For a functional transducer $\mathcal{T}$, we write $[\![\mathcal{T}]\!]$ the sequential function realized by $\mathcal{T}$, i.e. the mapping from input to output signals $[\![\mathcal{T}]\!] : w \mapsto u$ for all u, w such that $\mathcal{T}$ has a run over w inducing u.

Definition 5.2 (Product). Let $\langle S_k, \underline{s}_k, P_k, Q_k, X_k, \iota_k, \Delta_k, \lambda_k, \gamma_k, \mathcal{F}_k \rangle$ for $k = 1, 2$ be timed transducers. Their synchronous product is the timed transducer $\langle S, \underline{s}, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$ such that

- $S = S_1 \times S_2$;
- $\underline{s} = (\underline{s}_1, \underline{s}_2)$;
- $P = P_1 \cup P_2$;
- $Q = Q_1 \cup Q_2$;
- $X = X_1 \cup X_2$;
- $\iota(s_1, s_2) = \iota_1(s_1) \wedge \iota_2(s_2)$;
- The transition relation Δ consists of:
  - simultaneous transitions $((s_1, s_2), g, Z, (s_1', s_2'))$, where $(s_1, g_1, Z_1, s_1') \in \Delta_1$, $(s_2, g_2, Z_2, s_2') \in \Delta_2$, $g = g_1 \wedge g_2$ and $Z = Z_1 \cup Z_2$,
  - left-sided transitions: $((s_1, s_2), g \wedge \iota_2(s_2), Z, (s_1', s_2))$, where $(s_1, g, Z, s_1') \in \Delta_1$;
  - right-sided transitions: $((s_1, s_2), \iota_1(s_1) \wedge g, Z, (s_1, s_2'))$, where $(s_2, g, Z, s_2') \in \Delta_2$;
- The input labeling consists of:
  - state labels: $\lambda(s_1, s_2) = \lambda_1(s_1) \wedge \lambda_2(s_2)$,
  - transition labels: we let $\lambda((s_1, s_2), g, Z, (s_1', s_2')) = \lambda_1(s_1, g_1, Z_1, s_1') \wedge \lambda_2(s_2, g_2, Z_2, s_2')$ when simultaneous, $\lambda((s_1, s_2), g \wedge \iota_2(s_2), Z, (s_1', s_2)) = \lambda_1(s_1, g, Z, s_1') \wedge \lambda_2(s_2)$ when left-sided, and $\lambda((s_1, s_2), \iota_1(s_1) \wedge g, Z, (s_1, s_2')) = \lambda_1(s_1) \wedge \lambda_2(s_2, g, Z, s_2')$ when right-sided;
- The output labeling γ is constructed similarly as λ;
- $\mathcal{F}$ has Büchi conditions G for all $F \in \mathcal{F}_i$, $i = 1, 2$ constructed as follows:
  - for all $s_i \in Q_i \cap F$, states of the form (s_1, s_2) are in G,
  - for all $\delta \in \Delta_i \cap F$, simultaneous and one-sided transitions built from δ are in G.

Let $\mathcal{T}$ be a timed transducer with input variables P and output variables Q. The projection $\mathcal{T}|_R$ of a timed transducer $\mathcal{T}$ on a subset of variables $R \subset P \cup Q$ consists in removing variables $r \in (P \cup Q) \setminus R$ from state and transition labels α by replacing them with $\exists r.\alpha$, or equivalently $\alpha[\top/r] \vee \alpha[\bot/r]$. Let then $\mathcal{T}_1$ and $\mathcal{T}_2$ be two timed transducers, such that $\mathcal{T}_i$ has input variables P_i and output variables Q_i for $i = 1, 2$. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *sequential-composable* if $P_1 \cap Q_2 = \emptyset$ and $Q_1 = P_2$. In that case their *sequential composition*, denoted $\mathcal{T}_1 ; \mathcal{T}_2$, is defined as the projection onto $P_1 \cup Q_2$ of the synchronous product of $\mathcal{T}_1$ by $\mathcal{T}_2$. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *parallel-composable* when $(P_1 \cup Q_1) \cap Q_2 = \emptyset$ and $(P_2 \cup Q_2) \cap Q_1 = \emptyset$. In that case their *parallel composition*, denoted $\mathcal{T}_1 \| \mathcal{T}_2$, is simply the synchronous product of $\mathcal{T}_1$ by $\mathcal{T}_2$.

Proposition 5.3 (Composition of Transducers). *The following holds for all functional timed transducers $\mathcal{T}_1$ and $\mathcal{T}_2$.*

- *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are parallel-composable, then $[\![\mathcal{T}_1 \| \mathcal{T}_2]\!](w) = [\![\mathcal{T}_1]\!](w|_{P_1}) \| [\![\mathcal{T}_2]\!](w|_{P_2})$ for any signal w over $P_1 \cup P_2$.*
- *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are sequential-composable, then $[\![\mathcal{T}_1 ; \mathcal{T}_2]\!](w) = [\![\mathcal{T}_2]\!]([\![\mathcal{T}_1]\!](w))$ for any signal w over P_1.*

6 FROM MITL TO TIMED TRANSDUCERS

In this section, we propose a translation from MITL formulae to temporal testers, which are timed transducers computing the characteristic sequential function of the formula. We proceed to construct temporal testers for every operator used in the normal form of Section 4.2.3. The temporal tester $\mathcal{T}$ associated to a unary (resp. binary) operator f is a transducer with input variable p (resp. p_1 and p_2) and output variable q realizing the sequential function captured by $\square(q \leftrightarrow f(p))$ (resp. $\square(q \leftrightarrow f(p_1, p_2))$. We then show that such simple testers can be composed into testers for arbitrary MITL formulae, enabling us to recover known results regarding MITL satisfiability and model checking.

6.1 Temporal Testers for $\mathcal{S}$ and $\mathcal{U}$

6.1.1 Continuity and Expansion Rules. Before presenting our construction of testers for $\varphi_1 \mathcal{S} \varphi_2$ and $\varphi_1 \mathcal{U} \varphi_2$ formulae, we study the left- and right-continuity of characteristic signals of *since* and *until* formulae, and give an inductive characterization of the transduction function associated to these operators.

Definition 6.1 (Continuity). A temporal formula φ is said to be

(1) *left-continuous* when for all signals w and times t, $(w, t) \models \varphi$ implies that either $t = 0$ or there exists $t' < t$ such that for all $t'' \in (t', t)$, $(w, t'') \models \varphi$;
(2) *right-continuous* when for all signals w and times t, $(w, t) \models \varphi$ implies that there exists $t' > t$ such that for all $t'' \in (t, t')$, $(w, t'') \models \varphi$.

Lemma 6.2 (Continuity of Since and Until). *For every formulae φ_1 and φ_2, we have that*
(1) *$\varphi_1 \mathcal{S} \varphi_2$ is left-continuous;*
(2) *$\varphi_1 \mathcal{U} \varphi_2$ is right-continuous.*

Proof. The property does not depend on the structure of φ_1 and φ_2 but only their truth value, hence we can assume without loss of generality atomic formulae $\varphi_1 \equiv p_1$ and $\varphi_2 \equiv p_2$.

We first show the left-continuity of *since*. Let w be an arbitrary input signal and $u = \dot{u}_0 \cdot u_0^{r_0} \cdot \dot{u}_1 \cdot u_1^{r_1} \cdots = \chi^{p_1 \mathcal{S} p_2}(w)$. We show that for any $i \geq 1$, $\dot{u}_i = u_{i-1}$. There exist $t' < t_i$ such that p_2 is satisfied at t' and that p_1 holds continuously throughout the interval (t', t_i). It follows that $(w, t) \models p_1 \mathcal{S} p_2$ everywhere in (t', t_i) and, consequently $u_{i-1} = 1 = \dot{u}_i$. If $\dot{u}_i = 0$, there are two

possibilities: either p_2 was never true at any $t' \in [0, t_i)$, and hence u was false in the whole interval $(0, t_i)$; or for any $t'' \in [0, t_i)$ where p_2 was true, there is $t' \in (t'', t_i)$ where p_1 was false, implying that $p_1 \mathcal{S} p_2$ was not satisfied at (t', t_i) and $u_{i-1} = 0 = \dot{u}_i$.

We then show the right-continuity of *until*. Let w be an arbitrary input signal and $u = \dot{u}_0 \cdot u_0^{r_0} \cdot \dot{u}_1 \cdot u_1^{r_1} \cdots = \chi^{p_1 \mathcal{U} p_2}(w)$. We show that for any $i \geq 0$, $\dot{u}_i = u_i$. Assume first that $\dot{u}_i = 1$. There exists $t' > t_i$ such that p_2 is satisfied at t' and that p_1 holds continuously throughout the interval (t_i, t'). It follows that $(w, t) \models p_1 \mathcal{U} p_2$ everywhere in (t_i, t') and, consequently $u_i = 1 = \dot{u}_i$. If $\dot{u}_i = 0$, there are two possibilities: either p_2 never becomes true at any $t' > t_i$ and hence u is false in the whole open interval (t, ∞); or for any $t'' > t_i$ where p_2 is true there is $t' \in (t_i, t'')$ where p_1 does not hold which implies that $p_1 \mathcal{U} p_2$ is not satisfied at (t_i, t') and $u_i = 0 = \dot{u}_i$. □

Building up on the observation about left- and right-continuity of *since* and *until*, we identify a set of inductive relations that characterize the semantics of these two operators:

Lemma 6.3 (Expansion rules for Since). *Let $w = \dot{w}_0 \cdot w_0^{r_0} \cdot \dot{w}_1 \cdot w_1^{r_1} \cdots$ and $u = \chi^{p_1 \mathcal{S} p_2}(w)$. Then u is of the form $\dot{u}_0 \cdot u_0^{r_0} \cdot \dot{u}_1 \cdot u_1^{r_1} \cdots$ and its value can be inferred from the initial condition $\dot{u}_0 = 0$, the left-continuity $\dot{u}_{i+1} = u_i$, and the following rules valid for all $i \geq 0$:*

(1) *if $w_i \models \neg p_1$, then $u_i = 0$;*
(2) *if $w_i \models p_1 \wedge p_2$, then $u_i = 1$;*
(3) *if $w_i \models p_1 \wedge \neg p_2$, there are three possibilities:*
   (a) *if $\dot{w}_i \models \neg p_1 \wedge \neg p_2$, then $u_i = 0$*
   (b) *if $\dot{w}_i \models p_2$, then $u_i = 1$*
   (c) *if $\dot{w}_i \models p_1 \wedge \neg p_2$, then $u_i = \dot{u}_i$.*

Proof. The value of u at a point segment $\dot{u}_i$ is given as follows. For $i = 0$, we have $\dot{u}_0 = 0$ following the semantics of $p_1 \mathcal{S} p_2$ evaluated at time 0, whose satisfaction requires the existence of $t' < 0$ which is a contradiction. For $i \geq 1$, we have $\dot{u}_i = u_{i-1}$ by left-continuity of since. The value of u in the ith open segment is determined with respect to the values of p_1 and p_2 in the same segment w_i and at the preceding singular point $\dot{w}_i$. For any $t \in (t_i, t_{i+1})$ in the ith segment, we have:

Case 1. For any $t' < t$ which is in (t_i, t_{i+1}), p_1 does not hold, by definition, throughout (t', t), hence $(w, t) \not\models p_1 \mathcal{S} p_2$, that is $u_i = 0$;

Case 2. There exists $t' < t$ which is also in (t_i, t_{i+1}), where, by definition, p_2 holds at t' and p_1 holds continuously throughout (t', t). Hence $(w, t) \models p_1 \mathcal{S} p_2$ for all such t and $u_i = 1$;

Case 3a. Proposition p_1 is false at t_i and p_2 does not hold anywhere in the interval (t_i, t), which implies that $p_1 \mathcal{S} p_2$ is violated throughout (t_i, t_{i+1}) and $u_i = 0$;

Case 3b. Proposition p_2 is true at t_i and p_1 is continuously true during (t_i, t), implying that $p_1 \mathcal{S} p_2$ is satisfied at (t_i, t_{i+1}) and $u_i = 1$;

Case 3c. Proposition p_1 holds and p_2 remains false throughout $[t_i, t)$. Hence, $p_1 \mathcal{S} p_2$ holds at t if and only if there exists $t' \in [0, t_i)$ where p_2 holds, and p_1 remains true during (t', t_i), that is $p_1 \mathcal{S} p_2$ holds at t_i. This implies that $p_1 \mathcal{S} p_2$ is satisfied at (t_i, t_{i+1}) if and only if it is satisfied at t_i and $u_i = \dot{u}_i$.

For any combination of valuations of p_1 and p_2 at the ith singular point and the adjacent open segment, one of the above rules applies. □

Lemma 6.4 (Expansion rules for Until). *Let $w = \dot{w}_0 \cdot w_0^{r_0} \cdot \dot{w}_1 \cdot w_1^{r_1} \cdots$ and $u = \chi^{p_1 \mathcal{U} p_2}(w)$. Then u is of the form $\dot{u}_0 \cdot u_0^{r_0} \cdot \dot{u}_1 \cdot u_1^{r_1} \cdots$ and its value can be inferred from the right-continuity $\dot{u}_i = u_i$ and from the following rules valid for all $i \geq 0$:*

(1) *if $w_i \models \neg p_1$, then $u_i = 0$,*

(2) *if* $w_i \models p_1 \wedge p_2$, *then* $u_i = 1$
(3) *if* $w_i \models p_1 \wedge \neg p_2$, *then either* w_i *is the last segment in* w *and* $u_i = 0$, *or:*
  (a) *if* $\dot{w}_{i+1} \models \neg p_1 \wedge \neg p_2$, *then* $u_i = 0$
  (b) *if* $\dot{w}_{i+1} \models p_2$, *then* $u_i = 1$
  (c) *if* $\dot{w}_{i+1} \models p_1 \wedge \neg p_2$, *then* $u_i = \dot{u}_i$.

Proof. The value of u at the ith point segment is $\dot{u}_i = u_i$ by right-continuity of *until*. The value of u in the ith open segment is determined with respect to the values of p_1 and p_2 in that same segment w_i and the next singular point $\dot{w}_{i+1}$. It is not hard to see that the above rules cover all possible combinations of values for p_1 and p_2 in pairs w_i and $\dot{w}_{i+1}$ of open segments and their adjacent singular points. For any $t \in (t_i, t_{i+1})$ in the ith segment, we have

Case 1. For any $t' > t$ in (t_i, t_{i+1}), and, by definition, p_1 is violated throughout (t, t'), hence $(w, t) \not\models p_1 \,\mathcal{U}\, p_2$ and $u_i = 0$;

Case 2. There exists $t' > t$ in (t_i, t_{i+1}) such that, by definition, p_2 holds at t' and p_1 holds continuously throughout (t, t'). Hence $(w, t) \models p_1 \,\mathcal{U}\, p_2$ for all such t and $u_i = 1$;

Case 3a. By definition, p_1 is false at t_{i+1} and p_2 does not hold anywhere in the interval (t, t_{i+1}), which implies that $p_1 \,\mathcal{U}\, p_2$ is violated throughout (t_i, t_{i+1}) and $u_i = 0$;

Case 3b. Proposition p_2 is true at t_{i+1} and p_1 continuously holds during (t, t_{i+1}), implying that $p_1 \,\mathcal{U}\, p_2$ is satisfied at (t_i, t_{i+1}) and $u_i = 1$;

Case 3c. Proposition p_1 holds and p_2 remains false throughout $(t, t_{i+1}]$. Hence, $p_1 \,\mathcal{U}\, p_2$ holds at t if and only if there exists $t' > t_{i+1}$ where p_2 holds, and p_1 remains true during (t_{i+1}, t'), that is $p_1 \,\mathcal{U}\, p_2$ holds at t_{i+1}. This implies that $p_1 \,\mathcal{U}\, p_2$ is satisfied at (t_i, t_{i+1}) if and only if it is satisfied at t_{i+1} and $u_i = \dot{u}_{i+1}$.

The only remaining case when $w_i \models p_1 \wedge \neg p_2$ is the special case where w_i is the last segment in the signal w, that is, segment w_i is defined over the interval (t_i, ∞). In that case there is no $t' > t_i$ where q is true, and $u_i = 0$. □

6.1.2 Temporal Tester for $\mathcal{S}$. We are now ready to construct the temporal tester $\mathcal{T}_{\mathcal{S}}$, producing a signal q associated to the formula $p_1 \,\mathcal{S}\, p_2$. We let $\mathcal{T}_{\mathcal{S}} = \langle S, \underline{s}, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$, where

- $S = \{s_0, \ldots, s_3\}$,
- $P = \{p_1, p_2\}$ and $Q = \{q\}$,
- $X = \emptyset$,
- $\Delta = \{\delta_1, \ldots, \delta_{20}\}$,
- $\mathcal{F} = \emptyset$,

and ι, λ, γ, and $\delta_1, \ldots, \delta_{20}$ are given in Table 2 (location part) and Table 3 (edge part).

The temporal tester $\mathcal{T}_{\mathcal{S}}$, producing a signal q for the formula $p_1 \,\mathcal{S}\, p_2$, is depicted in Figure 6 and works as follows. Following Lemma 6.2, the output at time 0 (in initial transitions δ_1 to δ_3) is $\neg q$, irrespective of the initial input values. Lemma 6.2 also require that the output at any singular point agrees with the output during the preceding open segment. The temporal tester $\mathcal{T}_{\mathcal{S}}$ realizes this requirement by ensuring that for all transitions $\delta = (s, \top, \emptyset, s') \in \Delta$ coming from a non-initial state $s \neq \underline{s}$, we have $\gamma(\delta) = \gamma(s)$. When reading an open input segment w_i of w, the tester remains in one of its locations (s_0 to s_3) and generates the corresponding output segments u_i, according to the semantic rules from Lemma 6.3. In location s_0, the tester reads an open input segment that satisfies $p_1 \wedge p_2$ and outputs q, regardless of the preceding singular point input values. Similarly, in location s_2, the tester reads an open input segment that satisfies $\neg p_1$ and outputs $\neg q$, independent of the preceding singular point value. When the tester reads an open input segment that models $p_1 \wedge \neg p_2$, the tester can be either in location s_1 with output q, or in location s_3 with output $\neg q$, depending on the preceding input history. We can distinguish three possibilities:

Table 2. States of $\mathcal{T}_{\mathcal{S}}$, their input and output labels, and invariants.

i	s_i	$\lambda(s_i)$	$\gamma(s_i)$	$\iota(s_i)$
0	s_0	$p_1 \wedge p_2$	q	$\top$
1	s_1	$p_1 \wedge \neg p_2$	q	$\top$
2	s_2	$\neg p_1$	$\neg q$	$\top$
3	s_3	$p_1 \wedge \neg p_2$	$\neg q$	$\top$

Table 3. Edges of $\mathcal{T}_{\mathcal{S}}$ and their input and output labels.

i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$	i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$
1	$(\underline{s}, \top, \emptyset, s_0)$	$\top$	$\neg q$	11	$(s_1, \top, \emptyset, s_2)$	$\top$	q
2	$(\underline{s}, \top, \emptyset, s_1)$	p_2	$\neg q$	12	$(s_1, \top, \emptyset, s_3)$	$\neg p_1 \wedge \neg p_2$	q
3	$(\underline{s}, \top, \emptyset, s_2)$	$\top$	$\neg q$	13	$(s_2, \top, \emptyset, s_0)$	$\top$	$\neg q$
4	$(\underline{s}, \top, \emptyset, s_3)$	$\neg p_2$	$\neg q$	14	$(s_2, \top, \emptyset, s_1)$	p_2	$\neg q$
5	$(s_0, \top, \emptyset, s_0)$	$\neg p_1 \vee \neg p_2$	q	15	$(s_2, \top, \emptyset, s_2)$	p_1	$\neg q$
6	$(s_0, \top, \emptyset, s_1)$	$p_1 \vee p_2$	q	16	$(s_2, \top, \emptyset, s_3)$	$\neg p_2$	$\neg q$
7	$(s_0, \top, \emptyset, s_2)$	$\top$	q	17	$(s_3, \top, \emptyset, s_0)$	$\top$	$\neg q$
8	$(s_0, \top, \emptyset, s_3)$	$\neg p_1 \wedge \neg p_2$	q	18	$(s_3, \top, \emptyset, s_1)$	p_2	$\neg q$
9	$(s_1, \top, \emptyset, s_0)$	$\top$	q	19	$(s_3, \top, \emptyset, s_2)$	$\top$	$\neg q$
10	$(s_1, \top, \emptyset, s_1)$	p_2	q	20	$(s_3, \top, \emptyset, s_3)$	$\neg p_1 \wedge \neg p_2$	$\neg q$

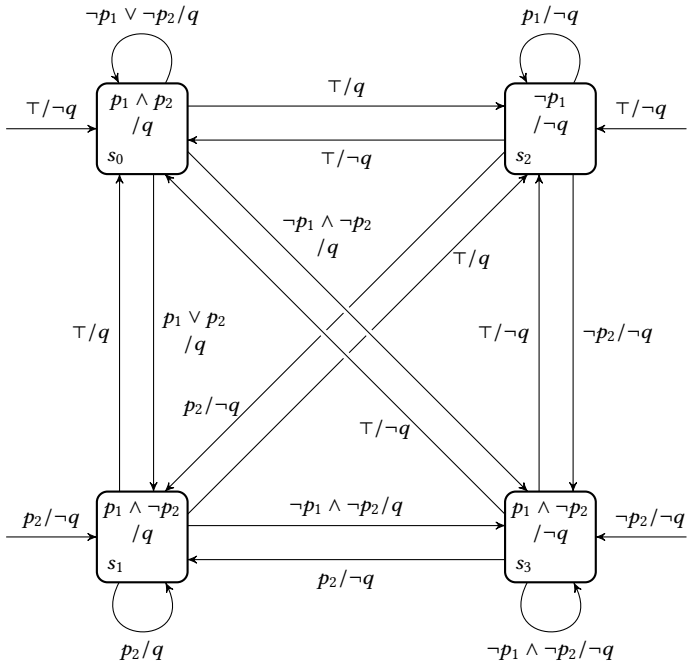

Fig. 6. Temporal tester $\mathcal{T}_{\mathcal{S}}$.

- The preceding singular input point satisfies $\neg p_1 \wedge \neg p_2$ and the tester is in location s_3 with output $\neg q$ (transitions δ_4, δ_8, δ_{16} and δ_{20});
- The preceding singular input point satisfies p_2 and the tester is in location s_1 with output q (transitions δ_6, δ_{10}, δ_{14} and δ_{18});
- The preceding singular point satisfies $p_1 \wedge \neg p_2$, the tester is in the location (s_1 or s_3) whose output agrees with the output at that singular point (transitions δ_5 and δ_{16}).

6.1.3 Temporal Tester for $\mathcal{U}$. We are now ready to construct the temporal tester $T_{\mathcal{U}}$ with output variable q and implementing the formula $p_1 \,\mathcal{U}\, p_2$. We let $T_q = \langle S, \underline{s}, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$ where

- $S = \{s_0, \ldots, s_3\}$,
- $P = \{p_1, p_2\}$ and $Q = \{q\}$,
- $X = \emptyset$,
- $\Delta = \{\delta_1, \ldots, \delta_{20}\}$,
- $\mathcal{F} = \{F_1\}$ where $F_1 = (S \cup \Delta) \setminus \{s_1\}$,

and ι, λ, γ, and $\delta_1, \ldots, \delta_{20}$ are given in Table 4 and Table 5. The temporal tester $\mathcal{T}_{\mathcal{U}}$ for operator *until* with input p_1, p_2 and output q is depicted in Figure 7. It is symmetric to the tester for *since* and can be obtained from its past counterpart by inverting the transition arrows. Unlike its past counterpart, which reads inputs, and determines the output according to the observed history, the tester for *until* predicts the output nondeterministically and has to either confirm or abort output predictions by future inputs.

Lemma 6.2 requires that all singular points have an output that agrees with the subsequent open output segment. The tester realizes this fact by letting the output on transitions be identical to the output of the target location.

When reading an open input segment $w_i^{r_i}$ of w, the tester remains in one of its locations (s_0 to s_3) and generates the corresponding output segment $u_i^{r_i}$ according to the semantic rules from Lemma 6.4.

Table 4. States of $\mathcal{T}_{\mathcal{U}}$, their input and output labels, and invariant.

i	s_i	$\lambda(s_i)$	$\gamma(s_i)$	$\iota(s_i)$
0	s_0	$p_1 \wedge p_2$	q	$\top$
1	s_1	$p_1 \wedge \neg p_2$	q	$\top$
2	s_2	$\neg p_1$	$\neg q$	$\top$
3	s_3	$p_1 \wedge \neg p_2$	$\neg q$	$\top$

Table 5. Edges of $\mathcal{T}_{\mathcal{U}}$ and their input and output labels.

i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$	i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$
1	$(\underline{s}, \top, \emptyset, s_0)$	$\top$	q	11	$(s_1, \top, \emptyset, s_2)$	p_2	$\neg q$
2	$(\underline{s}, \top, \emptyset, s_1)$	$\top$	q	12	$(s_1, \top, \emptyset, s_3)$	p_2	$\neg q$
3	$(\underline{s}, \top, \emptyset, s_2)$	$\top$	$\neg q$	13	$(s_2, \top, \emptyset, s_0)$	$\top$	q
4	$(\underline{s}, \top, \emptyset, s_3)$	$\top$	$\neg q$	14	$(s_2, \top, \emptyset, s_1)$	$\top$	q
5	$(s_0, \top, \emptyset, s_0)$	$\neg p_1 \vee \neg p_2$	q	15	$(s_2, \top, \emptyset, s_2)$	p_1	$\neg q$
6	$(s_0, \top, \emptyset, s_1)$	$\top$	q	16	$(s_2, \top, \emptyset, s_3)$	$\top$	$\neg q$
7	$(s_0, \top, \emptyset, s_2)$	$\top$	$\neg q$	17	$(s_3, \top, \emptyset, s_0)$	$\neg p_1 \wedge \neg p_2$	q
8	$(s_0, \top, \emptyset, s_3)$	$\top$	$\neg q$	18	$(s_3, \top, \emptyset, s_1)$	$\neg p_1 \wedge \neg p_2$	q
9	$(s_1, \top, \emptyset, s_0)$	$p_1 \vee p_2$	q	19	$(s_3, \top, \emptyset, s_2)$	$\neg p_2$	$\neg q$
10	$(s_1, \top, \emptyset, s_1)$	p_2	q	20	$(s_3, \top, \emptyset, s_3)$	$\neg p_1 \wedge \neg p_2$	$\neg q$

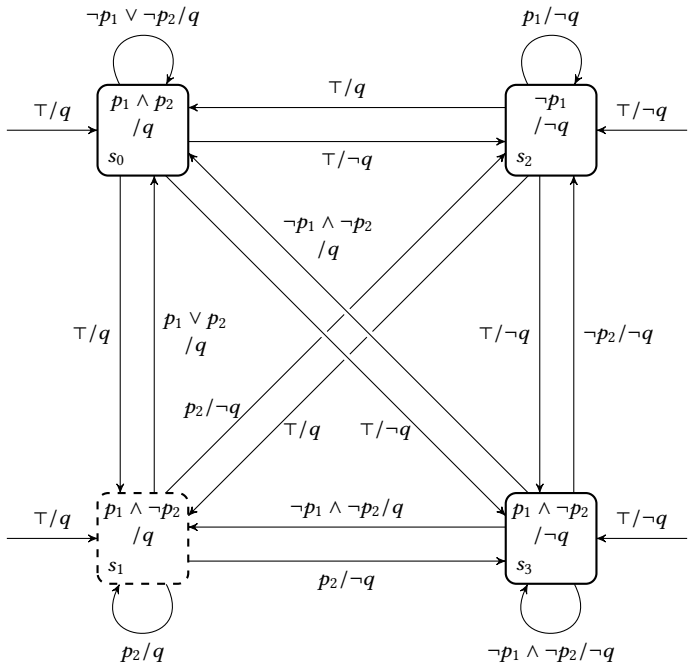

Fig. 7. Temporal tester $\mathcal{T}_{\mathcal{U}}$. Dashed line indicates that state s_1 does not belong to Büchi condition.

In locations s_0 and s_2, the tester, which is a transducer for formula $p_1 \,\mathcal{U}\, p_2$, behaves the same way as its past counterpart, and outputs q and $\neg q$, respectively, regardless of the input values in the next adjacent singular point.

When the tester reads an open input segment that satisfies $p_1 \wedge \neg p_2$, the tester can be either in location s_1 with output q, or in location s_3 with output $\neg q$, depending on the prediction that the tester made. The prediction is nondeterministic and results in two parallel runs of the tester, one of which will be aborted later. We can distinguish three possibilities:

- The tester predicts the output q (location s_1) that is followed by a subsequent singular point that satisfies p_2 (transitions δ_9, δ_{10}, δ_{11} and δ_{12});
- The tester predicts the output $\neg q$ (location s_3) that is followed by a subsequent singular point that satisfies $\neg(p_1 \vee p_2)$ (transitions δ_{17}, δ_{18}, δ_{19} and δ_{20});
- The tester predicts the output to be either q or $\neg q$ (location s_1 or s_3 and the following singular point satisfies $p_1 \wedge \neg p_2$. In this case, neither prediction can be immediately confirmed or aborted, and more input has to be read to reject the wrong prediction. However, the prediction made during the open segment that satisfies $p_1 \wedge \neg p_2$ has to agree with the output prediction at the adjacent $p_1 \wedge \neg p_2$ singular point, so the tester has to take the transition δ_9 from s_1, and the transition δ_{19} from s_3.

The only input signals that lead to two infinite runs are those that end with an infinite open segment that satisfies $p_1 \wedge \neg p_2$ and these signals violate the $p_1 \,\mathcal{U}\, p_2$ formula. To abort the wrong run that predicts q all along the segment, we forbid the tester to remain forever in s_1 without taking any transition, with $\mathcal{F} = (S \cup \Delta) \setminus \{s_1\}$.

We have shown transducers implementing the untimed *since* and *until* operators of MITL. The developments in this section demonstrate:

Proposition 6.5 (Correctness of $\mathcal{T}_{\mathcal{S}}$ and $\mathcal{T}_{\mathcal{U}}$). *For any signal w over variables p_1, p_2 and q,*

- $w \models \square(q \leftrightarrow (p_1 \,\mathcal{S}\, p_2))$ *iff* $[\![\mathcal{T}_{\mathcal{S}}]\!](w_{p_1p_2}) = w_q$;
- $w \models \square(q \leftrightarrow (p_1 \,\mathcal{U}\, p_2))$ *iff* $[\![\mathcal{T}_{\mathcal{U}}]\!](w_{p_1p_2}) = w_q$.

6.2 Temporal Testers for $\diamondsuit\!\!\!-_{(0,a)}$ and $\diamondsuit_{(0,a)}$

The temporal tester $\mathcal{T}_{\diamondsuit\!\!\!-_{(0,a)}}$ computing signal q representing the truth value of formula $\diamondsuit\!\!\!-_{(0,a)}\, p$ has to monitor the truth value of p and memorize with clock variables times where this value has changed. According to the memorized input history, the tester generates the correct output. The temporal tester $\mathcal{T}_{\diamondsuit_{(0,a)}}$ computing $q \leftrightarrow \diamondsuit_{(0,a)}\, p$ is similar to the one of $\diamondsuit\!\!\!-_{(0,a)}$, but unlike its past counterpart, it generates the output nondeterministically and later checks whether the actual input confirms such predictions, aborting the wrong ones.

6.2.1 Temporal Tester for $\diamondsuit\!\!\!-_{(0,a)}$. Consider the formula $\psi \equiv \diamondsuit\!\!\!-_{(0,a)}\, \varphi$ and let w be an arbitrary input signal. When φ holds in w for some interval I with endpoints t_i and t_j, then ψ is satisfied throughout the *open* interval $I \oplus (0,a) = (t_i, t_j + a)$, regardless of the shape of I among (t_i, t_j), $[t_i, t_j)$, $(t_i, t_j]$ or $[t_i, t_j]$. In other words, the satisfaction of ψ within the interval $(t_i, t_j + a)$ does not depend on the satisfaction of φ at t_i and t_j.

Let $I = (t_i, t_j)$ be an open segments where φ continuously holds. Following the fact that for every $t \in I$, there exists $t' < t$ such that $t' \in I$, it follows that ψ is also satisfied throughout I.

Let I be a segment with endpoints t_i and t_j such that φ is violated throughout I. The effect of the segments where φ is violated on the output depends on the actual duration of the "false" segment:

(1) if $t_j - t_i < a$, the false segment is too short and does not affect the output of the tester, i.e. $\diamondsuit\!\!\!-_{(0,a)}\, \varphi$ is satisfied throughout I. In fact, for all $t \in I$, there exists $t' \in t \ominus (0,a)$ such that $t' \leq t_i$ and where φ holds, hence ψ remains satisfied throughout I.
(2) if $t_j - t_i = a$, ψ remains satisfied throughout (t_i, t_j), but is violated at t_j.
(3) if $t_j - t_i > a$, then ψ remains satisfied during $(t_i, t_i + a)$, and violated throughout $[t_i + a, t_j]$.

These three cases are depicted in Figure 8.

We now construct the temporal tester $\mathcal{T}_{\diamondsuit\!\!\!-_{(0,a)}}$, a transducer realizing the relation $q \leftrightarrow \diamondsuit\!\!\!-_{(0,a)}\, p$, following the above observations. Let $\mathcal{T}_{\diamondsuit\!\!\!-_{(0,a)}} = \langle Q, s_0, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$, where

- $S = \{s_0, s_1, s_2\}$,
- $P = \{p\}$ and $Q = \{q\}$,
- $X = \{x\}$,
- $\Delta = \{\delta_1, \ldots, \delta_{12}\}$,
- $\mathcal{F} = \emptyset$,

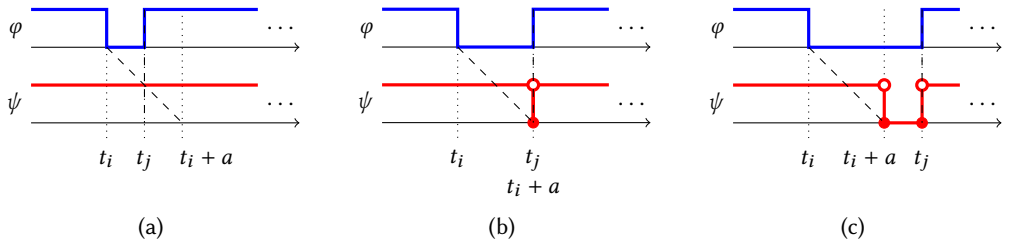

Fig. 8. Satisfaction of $\psi \equiv \diamondsuit\!\!\!-_{(0,a)}\, \varphi$ over time for three cases where φ is violated between t_i and t_j: (a) $t_j - t_i < a$; (b) $t_j - t_i = a$; (c) $t_j - t_i > a$.

Table 6. States of $\mathcal{T}_{\Diamond_{(0,a)}}$, their input and output labels, and invariant.

i	s_i	$\lambda(s_i)$	$\gamma(s_i)$	$\iota(s_i)$
0	s_0	p	q	$\top$
1	s_1	$\neg p$	q	$x < a$
2	s_2	$\neg p$	$\neg q$	$\top$

Table 7. Edges of $\mathcal{T}_{\Diamond_{(0,a)}}$ and their input and output labels.

i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$	i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$
1	$(\underline{s}, \top, \emptyset, s_0)$	$\top$	$\neg q$	7	$(s_1, x = a, \emptyset, s_0)$	$\top$	$\neg q$
2	$(\underline{s}, \top, \{x\}, s_1)$	p	$\neg q$	8	$(s_1, x < a, \{x\}, s_1)$	p	q
3	$(\underline{s}, \top, \emptyset, s_2)$	$\neg p$	$\neg q$	9	$(s_1, x = a, \{x\}, s_1)$	p	$\neg q$
4	$(s_0, \top, \emptyset, s_0)$	$\neg p$	q	10	$(s_1, x = a, \emptyset, s_2)$	$\neg p$	$\neg q$
5	$(s_0, \top, \{x\}, s_1)$	$\top$	q	11	$(s_2, \top, \emptyset, s_0)$	$\top$	$\neg q$
6	$(s_1, x < a, \emptyset, s_0)$	$\top$	q	12	$(s_2, \top, \{x\}, s_1)$	p	$\neg q$

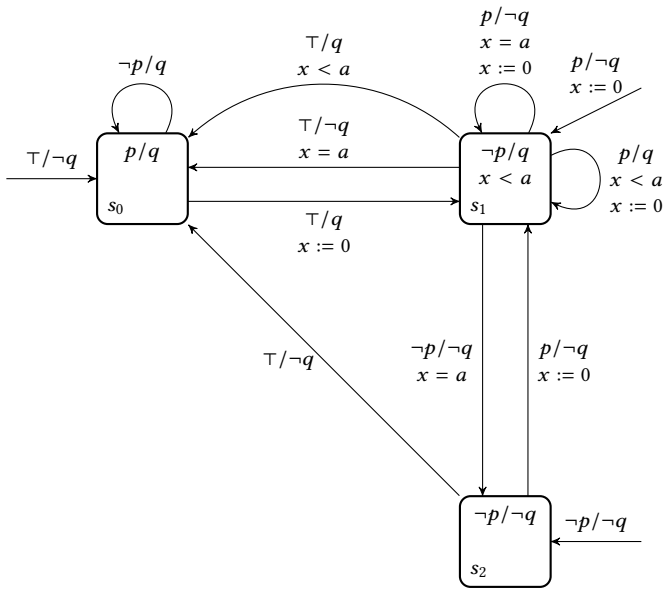

Fig. 9. Temporal tester $\mathcal{T}_{\Diamond_{(0,a)}}$.

and ι, λ, γ, and $\delta_1, \ldots, \delta_{12}$ are given in Table 6 and Table 7.

The temporal tester $\mathcal{T}_{\Diamond_{(0,a)}}$ with input p and output $q \leftrightarrow \Diamond_{(0,a)} p$ is depicted in Figure 9 and works as follows. The tester observes the input behavior w and moves through its locations, generating the correct output. At time $t = 0$, the output trivially satisfies $\neg q$. In location s_0, the tester reads an open input segment that satisfies p, hence it generates the output where q holds true. Singular points violating p are ignored, a fact which is reflected by transition δ_4. When the tester observes an open input segment switching to p being violated, it moves from s_0 to s_1 and

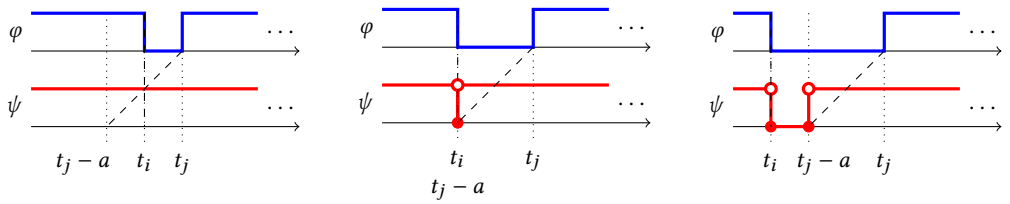

Fig. 10. Satisfaction of $\psi \equiv \Diamond_{(0,a)} \varphi$ over time for three cases where φ is violated between t_i and t_j: (a) $t_j - t_i < a$; (b) $t_j - t_i = a$; (c) $t_j - t_i > a$.

resets clock x (transition δ_5).[7] The clock x measures the distance from the last point in time when the input satisfied p. As long as the value of the clock is smaller than a, the tester outputs q as the property is satisfied. From location s_1, there are three possible continuations according to the relation between a and the duration of the segment that violates p:

(1) Proposition p becomes true before x reaches a, meaning that the segment that violates p was strictly smaller than a (transitions δ_6 and δ_8). Such "short" periods where p is violated are ignored by the tester, and the output continuously satisfies q.
(2) Proposition p becomes true when $x = a$. This situation is realized by transitions δ_7 and δ_9, resulting in a singular point where the output violates q.
(3) Proposition p is still violated when x reaches a and the tester moves to s_2. The output generated in s_2 violates q because the tester is in location s_2 if the last input that satisfied p happened more than a time ago. When the tester is in s_2 and observes an input that satisfies p, an appropriate transition is taken. In the case where the tester observes a singular point satisfying p, transition δ_{12} is taken, and transition δ_{11} is taken otherwise.

6.2.2 Temporal Tester for $\Diamond_{(0,a)}$. Consider the temporal formula $\psi \equiv \Diamond_{(0,a)} \varphi$ and let w be an arbitrary input signal. When φ holds in w for some interval I with endpoints t_i and t_j, then ψ is satisfied throughout the *open* interval $I \ominus (0, a) = (t_i - a, t_j)$. Symetrically to the case of past operators, this is regardless of the shape of I among (t_i, t_j), $[t_i, t_j)$, $(t_i, t_j]$ or $[t_i, t_j]$. The satisfaction of ψ within the interval $(t_i, t_j + a)$ does not depend on the satisfaction of φ at t_i and t_j. When φ becomes violated, there are three possible output behaviors depending on the actual duration of the segment where φ does not hold. To avoid repetition, we just illustrate these cases in Figure 10.

We now construct the temporal tester $\mathcal{T}_{\Diamond_{(0,a)}}$ implementing the temporal relation $q \leftrightarrow \Diamond_{(0,a)} p$, following the above observations. Let $\mathcal{T}_{\Diamond_{(0,a)}} = \langle S, \underline{s}, P, Q, X, \iota, \Delta, \lambda, \gamma, \mathcal{F} \rangle$, where

- $S = \{s_0, \ldots, s_3\}$,
- $P = \{p\}$ and $Q = \{q\}$,
- $X = \{x\}$,
- $\Delta = \{\delta_1, \ldots, \delta_{17}\}$,
- $\mathcal{F} = \emptyset$,

and ι, λ, γ, and $\delta_1, \ldots, \delta_{17}$ are given in Table 8 and Table 9. Over an open input segment where p holds the tester, depicted in Figure 11, is in location s_0 and the output satisfies q throughout that segment. It is not hard to see that for any t in such a segment, there exists some $t' > t$ that is also within the segment and such that p holds at t'. Singular points at which p is violated are

[7] We have seen that the tester does not need to distinguish whether the input satisfies or violates p at the moment of the transition.

Table 8. States of $\mathcal{T}_{\Diamond_{(0,a)}}$, their input and output labels, and invariant.

i	s_i	$\lambda(s_i)$	$\gamma(s_i)$	$\iota(s_i)$
0	s_0	p	q	$\top$
1	s_1	$\neg p$	q	$x < a$
2	s_2	$\neg p$	q	$x < a$
3	s_3	$\neg p$	$\neg q$	$\top$

Table 9. Edges of $\mathcal{T}_{\Diamond_{(0,a)}}$ and their input and output labels.

i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$	i	δ_i	$\lambda(\delta_i)$	$\gamma(\delta_i)$
1	$(\underline{s}, \top, \emptyset, s_0)$	$\top$	q	10	$(s_1, x = a, \{x\}, s_1)$	p	$\neg q$
2	$(\underline{s}, \top, \{x\}, s_1)$	$\top$	$\neg q$	11	$(s_1, x = a, \{x\}, s_2)$	p	q
3	$(\underline{s}, \top, \{x\}, s_2)$	$\top$	q	12	$(s_1, x = a, \emptyset, s_3)$	p	$\neg q$
4	$(\underline{s}, \top, \emptyset, s_3)$	$\top$	$\neg q$	13	$(s_2, x < a, \emptyset, s_0)$	$\top$	q
5	$(s_0, \top, \emptyset, s_0)$	$\neg p$	q	14	$(s_2, x < a, \{x\}, s_1)$	p	$\neg q$
6	$(s_0, \top, \{x\}, s_1)$	$\top$	$\neg q$	15	$(s_2, x < a, \{x\}, s_2)$	p	q
7	$(s_0, \top, \{x\}, s_2)$	$\top$	q	16	$(s_2, x < a, \emptyset, s_3)$	p	$\neg q$
8	$(s_0, x < a, \emptyset, s_3)$	$\top$	$\neg q$	17	$(s_3, \top, \{x\}, s_1)$	$\neg p$	$\neg q$
9	$(s_1, x = a, \emptyset, s_0)$	$\top$	q				

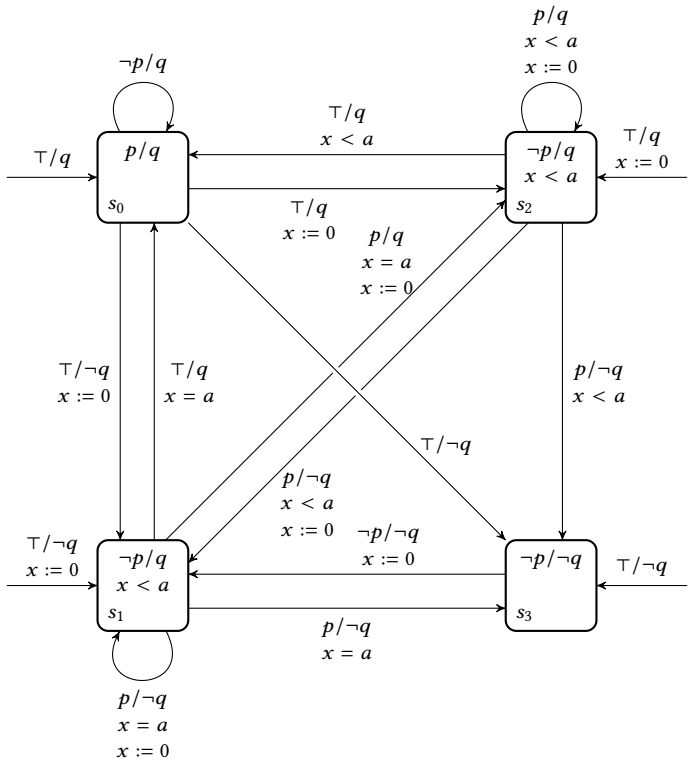

Fig. 11. Temporal tester $\mathcal{T}_{\Diamond_{(0,a)}}$.

ignored, this fact is reflected by transition δ_5. When the tester detects a switch to an input segment $I = (t_i, t_j)$ that violates p, it can make one of three output predictions, according to the duration of I. The tester can:

(1) Predict that $t_j - t_i < a$, a case realized by transition δ_7, which moves the tester into location s_2. In δ_7, the clock x is reset, ensuring that the delay until the next observation of an input satisfying p is strictly smaller than a, i.e. that one of δ_{13}, δ_{14}, δ_{15} or δ_{16} is taken by the tester. The run is aborted if p remains violated at $x = a$. The output at location s_2 satisfies q.
(2) Predict that $t_j - t_i = a$, a case realized by transition δ_6, which has s_1 as its outgoing location. Note that the output of δ_6 negates q while the output of s_1 satisfies it. Consequently, this prediction yields a singular point (time t_i) where q does not hold. The clock x is reset in δ_6 and is used to measure the delay until the next occurrence of an input satisfying p, which must occur when x is exactly equal to a (realized by transitions δ_9, δ_{10}, δ_{11} and δ_{12}). All other runs are aborted.
(3) Predict that $t_j - t_i > a$, by taking transition δ_8 to s_3. In s_3, the tester is only allowed to observe inputs that violate p, all other runs are aborted. The tester makes a nondeterministic guess of the time instant t such that $t_j - t = a$, and consequently takes transition δ_{17} to s_1 that resets the clock x, after which it behaves as in the previous case. Predicting transition δ_{17} at any time other than $t = t_j - a$ results in a run that is eventually aborted. Transition δ_{17} also marks the last point in time in that segment where q is violated.

We have shown transducers implementing timed *once* and *eventually* operators of MITL. The developments in this section demonstrate:

Proposition 6.6 (Correctness of $\mathcal{T}_{\diamondsuit\!\!\!-_{(0,a)}}$ and $\mathcal{T}_{\diamondsuit_{(0,a)}}$). *For any signal w over variables p and q,*

- $w \models \square(q \leftrightarrow \diamondsuit\!\!\!-_{(0,a)}\, p)$ *iff* $[\![\mathcal{T}_{\diamondsuit\!\!\!-_{(0,a)}}]\!](w_p) = w_q$;
- $w \models \square(q \leftrightarrow \diamondsuit_{(0,a)}\, p)$ *iff* $[\![\mathcal{T}_{\diamondsuit_{(0,a)}}]\!](w_p) = w_q$.

6.3 Main Result

Let us take the following definitions:

- The *size* of an MITL formula φ is the number of its subformulae.
- The *resolution* of φ is the maximal relative interval width in φ, defined by $r(p) = 0, r(\varphi_1 \vee \varphi_2) = \max\{r(\varphi_1), r(\varphi_2)\}$, $r(\neg\varphi) = r(\varphi)$, and $r(\varphi_1\, \mathcal{U}_I\, \varphi_2) = r(\varphi_1\, \mathcal{S}_I\, \varphi_2) = \max\{r(\varphi_1), r(\varphi_2), r(I)\}$, where $r(I) = 2\left\lceil \frac{\sup I}{\sup I - \inf I} \right\rceil + 1$ if $\sup I < \infty$, $r(I) = 1$ otherwise.

We now state and prove the main result of this paper.

Theorem 6.7 (MITL Testers). *For any MITL formula of size m and resolution n, one can construct a temporal tester with $O(mn)$ clocks and $2^{O(mn)}$ locations.*

Proof. Let φ be an arbitrary MITL formula with size m and resolution n. We begin by rewriting φ into the equivalent formula φ' according to the normal form of Proposition 4.5. Formula φ' has size in $O(mn)$, and resolution 1.

The construction of a tester $\mathcal{T}^q_{\varphi'}$ for normal form MITL formula φ' with input variables P and fresh output variable q is inductive on the structure of φ'. For each subformula ψ of φ', we construct

a tester for its main subformula(s), and compose it with a tester associated to its main operator:

$$\mathcal{T}^q_{\neg\psi} = \mathcal{T}^p_\psi \,;\mathcal{T}^q_{\neg p} \qquad \mathcal{T}^q_{\psi_1\vee\psi_2} = (\mathcal{T}^{p_1}_{\psi_1} \| \mathcal{T}^{p_2}_{\psi_2}) \,;\mathcal{T}^q_{p_1\vee p_2}$$
$$\mathcal{T}^q_{\Diamond_{(0,a)}\psi} = \mathcal{T}^p_\psi \,;\mathcal{T}^q_{\Diamond_{(0,a)} p} \qquad \mathcal{T}^q_{\psi_1\,\mathcal{U}\,\psi_2} = (\mathcal{T}^{p_1}_{\psi_1} \| \mathcal{T}^{p_2}_{\psi_2}) \,;\mathcal{T}^q_{p_1\,\mathcal{U}\,p_2}$$
$$\mathcal{T}^q_{\Diamond\!\!\!-_{(0,a)}\psi} = \mathcal{T}^p_\psi \,;\mathcal{T}^q_{\Diamond\!\!\!-_{(0,a)} p} \qquad \mathcal{T}^q_{\psi_1\,\mathcal{S}\,\psi_2} = (\mathcal{T}^{p_1}_{\psi_1} \| \mathcal{T}^{p_2}_{\psi_2}) \,;\mathcal{T}^q_{p_1\,\mathcal{S}\,p_2}$$

where p, p_1 and p_2 are fresh variables. Testers for atomic and propositional formulae λ can be built as two-state components with a complete transition graph whose locations and transitions are labeled with λ/q and $\neg\lambda/\neg q$. We then let $\mathcal{T}^q_\varphi = \mathcal{T}^q_{\varphi'}$.

The correctness of the above construction follows directly from propositions 6.5 and 6.6. lemmas 6.2, 6.3 and 6.4. Every atomic tester built to translate ψ from subsections 6.1 and 6.2 has at most 4 locations and 1 clock; testers for Boolean or propositional variables have 2 locations and do not have clocks. Thus, in the product of all atomic testers used to build the tester for φ there are $2^{O(mn)}$ locations and $O(mn)$ clocks. □

Let us remark that the rewritings involved in Proposition 4.5 do not incur an additional cost in terms of clock-complexity. The developments in [29] show that exactly $2\lceil\frac{a}{b-a}\rceil+1$ clocks are needed to realize a tester for $\Diamond_{(a,b)}$. The number of clocks required for past operators is the same, as showed in [53]. The testers $\mathcal{T}_{\Diamond_{(a,b)}}$ and $\mathcal{T}_{\Diamond_{(a,b)}}$ obtained for an arbitrary interval (a,b) by repeated applications of Lemma 4.3 are indeed exactly optimal in their number of clocks. For the case of (semi-)closed intervals this necessitates that multiple occurrences of the same *timed eventually* formula share the same tester. This is one of the benefits of our compositional translation: several occurrences of a subformula can share the same tester.

We can check that Theorem 6.7 allows us to recover the results of [5]:

Corollary 6.8 (MITL Verification). *The satisfiability of MITL and the model-checking of timed automata against MITL are decidable in EXPSPACE.*

Proof. Let ψ be an MITL formula of length $|\varphi|$ in binary notation. The resolution m of φ is in $2^{O(|\varphi|)}$ and its size n in $O(|\varphi|)$. We can construct a tester $\mathcal{T}_\varphi$ for φ according to Theorem 6.7, and take its sequential composition with an acceptor for q to obtain a acceptor $\mathcal{A}_\varphi$ for φ whose dimensions don't exceed that of $\mathcal{T}_\varphi$.

Using a dag representation, the normal form of φ can be computed in polynomial time. The emptiness problem for (event-based) timed automata is in PSPACE by reduction to a path problem in its *region graph* [4]. This result can easily be adapted to signal-based timed automata with rational time constants: the signal-based semantics can be encoded e.g. using urgent locations, and rational time constants can be accomodated by scaling them into integers (after scaling, the length of time constants remains polynomial in $|\varphi|$). A node in the region graph can be described in space logarithmic in the number of location and polynomial in the number of clocks and the total length of time constants. When applying the procedure of [4], the acceptor $\mathcal{A}_\varphi$ does not require to be constructed explicitly. The path problem in the region graph can be solved nondeterministically by checking the existence of an edge between two nodes, while storing a constant number of nodes. The region graph of $\mathcal{A}_\varphi$ is the product of the region graphs of its tester components, so that nodes and edges can be computed on the fly. The description of each node and edge has a length exponential in $|\varphi|$, so that the satisfiability of MITL is in EXPSPACE.

For model-checking a (signal-based) timed automaton $\mathcal{B}$ against φ, it suffices to replace $\mathcal{A}_\varphi$ with $\mathcal{B}\|\mathcal{A}_{\neg\varphi}$ in the above. The resulting decision procedure uses space polynomial in $|\mathcal{B}|$ and exponential in $|\varphi|$, so that MITL timed model-checking is in EXPSPACE. □

The problems considered in Corollary 6.8 are in fact EXPSPACE-complete [5].

7 CONCLUSION

We presented a compositional method based on temporal testers for translating MITL specifications in their full generality to timed automata. Modularity, which lies in the heart of our approach, yields a surprisingly simple and elegant translation, especially when compared to previous work. The essence of this work is:

(1) We demonstrate that four basic operators, $\mathcal{U}$, $\mathcal{S}$, $\diamondsuit_{(0,a)}$ and $\diamondsuit_{(0,a)}$, are sufficient to express full MITL;
(2) We construct the timed testers for these operators. These testers are straightforward to understand and implement, each consisting of at most four locations and a single clock;
(3) We show that a network of communicating testers, derived from the structure of an MITL property, yields an equivalent timed automaton.

Apart from the theoretical contributions of this paper, we also see several more practical applications of our translation. Firstly, it may provide an alternative automaton-based procedure for runtime monitoring [55]. In fact, the offline monitoring procedure for MITL/STL described in [52, 64] is based on the very similar concept of *satisfaction signals* propagated between sub-formulae. After translating specifications to automata, we can check membership of a given signal by applying on-the-fly subset construction [50] and detect property violations in an online fashion. Secondly, we can use the timed automaton resulting from an MITL specification to generate test cases that explore the requirements while achieving a given structural coverage.

REFERENCES

[1] IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005) – IEEE Standard for Property Specification Language (PSL). Standard, IEEE, 2010.
[2] ANSI/IEEE 1800-2012 – IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language. Standard, 2012.
[3] Rajeev Alur. Timed automata. In *International Conference on Computer Aided Verification*, pages 8–22. Springer, 1999.
[4] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.
[5] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
[6] Rajeev Alur and Thomas A Henzinger. Back to the future: towards a theory of timed regular languages. In *Foundations of Computer Science*, pages 177–186. IEEE, 1992.
[7] Rajeev Alur and Thomas A Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, pages 74–106. Springer, 1992.
[8] Rajeev Alur and Thomas A Henzinger. A really temporal logic. *Journal of the ACM (JACM)*, 41(1):181–203, 1994.
[9] Eugene Asarin. Challenges in timed languages: From applied theory to basic theory. *The Bulletin of the European Association for Theoretical Computer Science*, 83:106–120, 2004.
[10] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.
[11] Eugene Asarin and Cătălin Dima. Balanced timed regular expressions1. *Electronic Notes in Theoretical Computer Science*, 68(5), 2003.
[12] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.
[13] David Basin, Srđan Krstić, and Dmitriy Traytel. Almost event-rate independent monitoring of metric dynamic logic. In *Runtime Verification*, pages 85–102. Springer, 2017.
[14] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media, 2013.
[15] Marcello M Bersani, Matteo Rossi, and Pierluigi San Pietro. An SMT-based approach to satisfiability checking of MITL. *Information and Computation*, 245:72–97, 2015.
[16] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, Joël Ouaknine, and James Worrell. Model checking real-time systems. In Clarke et al. [28], chapter 29, pages 1001 – 1046.

[17] Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata. In *Formal Modeling and Analysis of Timed Systems*, pages 47–61, 2013.

[18] Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata over infinite words. In *Formal Modeling and Analysis of Timed Systems*, pages 69–84, 2014.

[19] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. Mightyl: A compositional translation from MITL to timed automata. In *Computer Aided Verification*, pages 421–440, 2017.

[20] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. Timed-automata-based verification of MITL over signals. In *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, pages 7:1–7:19, 2017.

[21] Janusz A. Brzozowski and Ernst Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, 10(1):19–35, 1980.

[22] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and computation*, 98(2):142–170, 1992.

[23] Ashok K Chandra and Larry J Stockmeyer. Alternation. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 98–108. IEEE, 1976.

[24] Alessandro Cimatti, Marco Roveri, Simone Semprini, and Stefano Tonetta. From PSL to NBA: a modular symbolic encoding. In *Formal Methods in Computer Aided Design, 2006. FMCAD'06*, pages 125–133. IEEE, 2006.

[25] Edmund Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another look at LTL model checking. In *International Conference on Computer Aided Verification*, pages 415–427. Springer, 1994.

[26] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[27] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 1999.

[28] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer International Publishing, 2018.

[29] Deepak D'Souza and R Matteplackel. A clock-optimal hierarchical monitoring automaton construction for MITL. Technical report, 2013.

[30] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 68–83. Springer, 2004.

[31] Cindy Eisner and Dana Fisman. Functional specification of hardware via temporal logic. In Clarke et al. [28], chapter 24, pages 795–829.

[32] Thomas Ferrère, Oded Maler, and Dejan Ničković. Trace diagnostics using temporal implicants. In *International Symposium on Automated Technology for Verification and Analysis*, pages 241–258. Springer, 2015.

[33] Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In *International Conference on Computer Aided Verification*, pages 53–65. Springer, 2001.

[34] Rob Gerth, Doron Peled, Moshe Y Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV*, pages 3–18. Springer, 1995.

[35] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to büchi automata. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 308–326. Springer, 2002.

[36] Thomas A Henzinger. It's about time: Real-time logics reviewed. In *International Conference on Concurrency Theory*, pages 439–454. Springer, 1998.

[37] Thomas A Henzinger, J-F Raskin, and P-Y Schobbens. The regular real-time languages. In *Automata, Languages and Programming*, pages 580–591. Springer, 1998.

[38] Yoram Hirshfeld and Alexander Rabinovich. Quantitative temporal logic. In *International Workshop on Computer Science Logic*, pages 172–187. Springer, 1999.

[39] Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[40] Yoram Hirshfeld and Alexander Rabinovich. Timer formulas and decidable metric temporal logic. *Information and Computation*, 198(2):148–178, 2005.

[41] Yoram Hirshfeld and Alexander Rabinovich. An expressive temporal logic for real time. In *Mathematical Foundations of Computer Science 2006*, pages 492–504. Springer, 2006.

[42] Yoram Hirshfeld and Alexander Rabinovich. Expressiveness of metric modalities for continuous time. In *Computer Science–Theory and Applications*, pages 211–220. Springer, 2006.

[43] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[44] Yonit Kesten and Amir Pnueli. A compositional approach to CTL* verification. *Theoretical Computer Science*, 331(2-3):397–428, 2005.

[45] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic verification of linear temporal logic specifications. In *International Colloquium on Automata, Languages, and Programming*, pages 1–16. Springer, 1998.

[46] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Bounded model checking of an MITL fragment for timed automata. In *13th International Conference on Application of Concurrency to System Design, ACSD 2013*, pages 216–225, 2013.

[47] Dileep Raghunath Kini, Shankara Narayanan Krishna, and Paritosh K. Pandya. On construction of safety signal automata for MITL[U,S] using temporal projections. In *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011*, pages 225–239, 2011.

[48] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[49] Ron Koymans, Jan Vytopil, and Willem P de Roever. Real-time programming and asynchronous message passing. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 187–197. ACM, 1983.

[50] Moez Krichen and Stavros Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[51] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[52] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.

[53] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real time temporal logic: Past, present, future. In *Formal Modeling and Analysis of Timed Systems*, pages 2–16. Springer, 2005.

[54] Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In *Formal Modeling and Analysis of Timed Systems*, pages 274–289. Springer, 2006.

[55] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, pages 475–505, 2008.

[56] Oded Maler and Amir Pnueli. On recognizable timed languages. In *International Conference on Foundations of Software Science and Computation Structures*, pages 348–362. Springer, 2004.

[57] Zohar Manna and Amir Pnueli. The anchored version of the temporal framework. In *Workshop/School/Symposium of the REX Project*, pages 201–284. Springer, 1988.

[58] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer Science & Business Media, 2012.

[59] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer Science & Business Media, 2012.

[60] Nicolas Markey. Temporal logic with past is exponentially more succinct. *EATCS Bulletin*, 79:122–128, 2003.

[61] Max Michel. Algebre de machines et logique temporelle. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 287–298. Springer, 1984.

[62] Max Michel. Computation of temporal operators. *Logique et Analyse*, 28(110/111):137–152, 1985.

[63] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32(3):321–330, 1984.

[64] Dejan Nickovic. *Checking timed and hybrid properties: Theory and applications*. PhD thesis, Université Joseph Fourier, Grenoble, France, 2008.

[65] Maurice Nivat and Dominique Perrin. Ensembles reconnaissables de mots bi-infinis. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 47–59. ACM, 1982.

[66] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Logic in Computer Science*, pages 188–197. IEEE, 2005.

[67] Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In *International Conference on Automated Deduction*, pages 748–752. Springer, 1992.

[68] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In *International Symposium on Formal Methods*, pages 573–586. Springer, 2006.

[69] Amir Pnueli and Aleksandr Zaks. On the merits of temporal testers. In *25 Years of Model Checking*, pages 172–195. Springer, 2008.

[70] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[71] Jean-François Raskin and Pierre-Yves Schobbens. State clock logic: A decidable real-time logic. In *International Workshop on Hybrid and Real-Time Systems*, pages 33–47. Springer, 1997.

[72] Nima Roohi and Mahesh Viswanathan. Revisiting mitl to fix decision procedures. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 474–494. Springer, 2018.

[73] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *International Conference on Computer Aided Verification*, pages 248–263. Springer, 2000.

[74] Boris A Trakhtenbrot. Understanding basic automata theory in the continuous time setting. *Fundamenta Informaticae*, 62(1):69–121, 2004.

[75] Moshe Y Vardi. Alternating automata and program verification. In *Computer Science Today*, pages 471–485. Springer, 1995.

[76] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

[77] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 694–715. Springer, 1994.