Symbolic Model Checking for Real-time Systems*

Thomas A. Henzinger†
Computer Science Department, Cornell University
Ithaca, NY 14853, U.S.A.

Xavier Nicollin‡ Joseph Sifakis‡ Sergio Yovine‡
VERIMAG, Miniparc-ZIRST rue Lavoisier
38330 Montbonnot Saint Martin, France

Abstract. We describe finite-state programs over real-numbered time in a guarded-command language with real-valued clocks or, equivalently, as finite automata with real-valued clocks. Model checking answers the question which states of a real-time program satisfy a branching-time specification (given in an extension of CTL with clock variables). We develop an algorithm that computes this set of states symbolically as a fixpoint of a functional on state predicates, without constructing the state space.

For this purpose, we introduce a μ-calculus on computation trees over real-numbered time. Unfortunately, many standard program properties, such as response for all nonzeno execution sequences (during which time diverges), cannot be characterized by fixpoints: we show that the expressiveness of the timed μ-calculus is incomparable to the expressiveness of timed CTL. Fortunately, this result does not impair the symbolic verification of "implementable" real-time programs—those whose safety constraints are machine-closed with respect to diverging time and whose fairness constraints are restricted to finite upper bounds on clock values. All timed CTL properties of such programs are shown to be computable as finitely approximable fixpoints in a simple decidable theory.

1 Introduction

Model checking is a powerful technique for the automatic verification of finite-state systems. Model-checking algorithms determine the states that satisfy a modal formula by a graph-theoretic analysis of the state space (Kripke structure) [QS81, EC82, CES86]. The main practical limitation of model-checking algorithms is caused by the size of the state graph, which grows exponentially with the number of parallel components in a system. One approach to confine this "state explosion problem" relies on the *symbolic* (rather than *enumerative*) representation of sets of states and computes the set that satisfies a formula as a fixpoint of a functional on state predicates. While the

*This paper will appear in *Information and Computation.* A preliminary version appeared in the proceedings of the *Seventh Annual IEEE Symposium on Logic in Computer Science* (LICS 1992), pp. 394–406.

†Partially supported by the National Science Foundation under grant CCR-9200794 and by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056.

‡Partially supported by the Esprit Basic Research Action REACT.

theoretical possibility of symbolic model checking by computing fixpoints was realized early [EC80, Sif82], the method has become practical only recently through the symbolic representation of state sets by binary decision diagrams (BDDs) [Bry86]. BDDs were first used for verification purposes by [CBM89] and for model checking by [BCM+92]. By now implementations of symbolic model-checking techniques have reported spectacular successes, in particular in the area of hardware verification [McM93].

The history of model checking has been considerably shorter in the case of real-time systems. First researchers focused on discrete time (the integers), for which the untimed model-checking methods can be readily extended [EMSS90, AH92b, Eme92]. New complexities arise if we insist that for the compositional modeling of asynchronous systems, time should not be discretized [Alu91]. We consider dense time (the reals). A standard dense-time approach models a real-time system as a transition relation together with a finite set of real-valued clocks that proceed at a uniform rate and constrain the times at which transitions may occur [AD90, Lew90, AFH91, AH92a, NSY93]. Only recently a graph-theoretic model-checking algorithm was found for these systems [ACD90]. Since the time component causes the state space to be infinite, the algorithm depends on a clever construction of a finite quotient of the infinite state graph. The size of this "region graph" of equivalence classes of states grows exponentially not only with the number of components in a system but also with the largest time constant and the number of clocks that are used to specify timing constraints. Thus the need for an alternative, "symbolic" approach to model checking, which avoids the explicit construction of the region graph, is particularly pressing in the real-time case.

The main problem in devising such a method lies in the definition of a proper next-state relation whose iteration allows us to compute all program properties we wish to consider. Since our time domain is dense, the next-state relation must not force time to advance by more than an infinitesimal amount. Its iteration, however, must force time to advance beyond any bound. This is because upper-bound clock constraints may restrict the time that a system can spend in a particular set of states, say, a loop. While the system may loop any finite number of times before the time bound expires, it cannot do so infinitely often. In other words, every upper-bound constraint hides a fairness condition.

Both requirements on the next-state relation seem and, in a precise technical sense we will discuss, generally are contradictory. We show that this result has, fortunately, little relevance to the verification of concrete real-time programs, including loops with upper bounds. We define the class of *divergence-safe* systems, for which inevitability is equivalent to time-bounded inevitability: for any event of a divergence-safe system there is a time bound such that if the event need not occur before that bound, then it need not occur ever. The class of divergence-safe systems contains all systems without fairness constraints other than those induced by constant upper bounds, which—it may be argued—is the only "implementable" notion of fairness. In the case of divergence-safe systems, we show that we can settle for a next-state relation that does not force time to advance ever. Such a relation allows us to compute possibility ($\exists\Diamond$) and its dual, invariance ($\forall\Box$), directly. Inevitability ($\forall\Diamond$) is reducible to time-bounded inevitability, which is but an invariance: time may not progress beyond the upper bound of any event without the event occurring.

We have now informally sketched our course of action for computing bounded and unbounded properties of real-time systems as fixpoints of functionals that are based on a next-state relation. Formally, we introduce a timed μ-calculus, Tμ, that is interpreted over timed computation trees. We compare this fixpoint calculus to standard real-time extensions of branching-time logics and find their expressive powers to be incomparable: the basic operator $\forall\Diamond$ of branching-time logics cannot be characterized by fixpoints in Tμ. However, as hinted above, we are able to give a translation from CTL with clock variables (TCTL of [Alu91]) to Tμ that agrees on a class of well-behaved divergence-safe models. This translation forms the basis of a symbolic model-checking procedure.

We apply this theory to a concrete real-time programming language. Real-time systems are defined in a guarded-command language with clocks, which is equivalent to *timed safety automata*—timed automata [AD90] without acceptance conditions. Every guarded-command real-time program defines a divergence-safe real-time system for which all formulas of Tμ and TCTL can be verified symbolically by computing fixpoints (indeed, finitely approximable fixpoints) of appropriate functionals. The practical computation of these functionals rests on our ability to express, for any given program, the next-state relation symbolically. Indeed, the extraction of the next-state relation from a program turns out to be rather nontrivial in the case that the program is not *nonzeno* [AL92, Hen92] (not machine-closed with respect to the divergence of time); that is, if it may prevent time from diverging. We show how a symbolic fixpoint approach can be used to test if a guarded-command real-time program (or a timed safety automaton) is nonzeno and, if not, to convert it into an equivalent nonzeno program.

We wish to conclude this introduction by pointing out that there is, of course, nothing "magical" about symbolic methods. Timed model checking is intrinsically difficult (PSPACE-hard) and already the known graph-theoretic algorithm on the exponential region graph is worst-case near-optimal [ACD90]. In practice, however, the "intuitive complexity" [BCM+92] of the state space is typically much smaller than the region graph. Only a symbolic method can exploit this phenomenon, by representing unions of regions symbolically as state predicates. Our model-checking algorithm constructs (the equivalent of) the full region graph but in extreme cases; typically it works on a quotient of the region graph that depends on the formula being checked. Moreover, symbolic methods can be applied to the verification of systems with an infinite state space even if it is not known *a priori* that there is a suitable finite quotient of the state space. While our symbolic model-checking procedure will fail to terminate if no such quotient exists, the procedure remains sound and led recently to the verification of systems with very general state spaces [ACHH93, NOSY93].

The remainder of the paper is organized as follows. In the next section, we define our model of real-time systems. Section 3 presents a guarded-command language and an automata-based language for the description of real-time systems. In Section 4, we review the branching-time logic TCTL and introduce the timed μ-calculus Tμ. Section 5 compares the expressive power of both specification languages. The definition of TCTL-formulas as fixpoints leads, in Section 6, to a symbolic model-checking algorithm for both Tμ and TCTL.

2 A Dense-time Model for Real-time Systems

We present a branching dense-time semantics for real-time systems. Our semantics integrates elements from several models that have been proposed in the literature; we are particularly indebted to the clocks of [AD90], the dense trees of [ACD90], the transition-delay dichotomy of [HMP91] and [NSY93], and the relative safety of [Hen92].

2.1 State trajectories

We model time as the nonnegative real numbers R^+. The state of a system is determined by the values of a finite set P of boolean variables (*propositions*), representing data and control, and by the values of a finite set C of real-valued variables (*clocks*). The clocks allow the system to make time-dependent decisions.

Definition 2.1 (state) A *state* σ is an interpretation of all propositions and clocks; that is, σ assigns to each proposition $p \in P$ a boolean value $\sigma(p) \in \{true, false\}$ and to each clock $x \in C$ a nonnegative real $\sigma(x) \in \mathsf{R}^+$. We write Σ for the set of all states. ■

For a delay $\delta \in \mathsf{R}^+$, let $\sigma+\delta$ denote the state that agrees with the state $\sigma \in \Sigma$ on all propositions and assigns the value $\sigma(x)+\delta$ to each clock $x \in C$. Given a set $A = \{v_1 := a_1, \ldots, v_n := a_n\}$ of variables $v_i \in P \cup C$ and corresponding values $a_i \in \{\mathit{true}, \mathit{false}\} \cup \mathsf{R}^+$, we write $\sigma[A]$ for the state that assigns the value a_i to the variable v_i for all $1 \leq i \leq n$ and agrees with the state $\sigma \in \Sigma$ on all other propositions and clocks.

The execution of a system generates an infinite succession of states. In each state the system may either take a transition or let time pass. With each transition, the system changes its data and control and resets some of the clocks; that is, the values of propositions are modified and certain clock values are reset to 0. Whenever time passes, all clock values increase uniformly with the rate of time. It follows that in any state the value of a clock x is equal to the amount of time that has elapsed since the last time x was reset.

Definition 2.2 (trajectory and divergence) A (*real-time*) *trajectory*

$$\bar{\sigma} \;=\; \sigma_0 \xrightarrow{\delta_0} \sigma_1 \xrightarrow{\delta_1} \sigma_2 \xrightarrow{\delta_2} \sigma_3 \xrightarrow{\delta_3} \cdots$$

consists of an infinite sequence of states $\sigma_i \in \Sigma$ and an infinite sequence of delays $\delta_i \in \mathsf{R}^+$ such that for all nonnegative integers i and all clocks $x \in C$, either $\sigma_{i+1}(x) = 0$ or $\sigma_{i+1}(x) = \sigma_i(x) + \delta_i$. A *position* of the trajectory $\bar{\sigma}$ is a pair (i, δ) consisting of a nonnegative integer i and a nonnegative real $\delta \leq \delta_i$. The *state at position* (i,δ) of $\bar{\sigma}$ is $\bar{\sigma}(i,\delta) = \sigma_i + \delta$. The *time at position* (i,δ) of $\bar{\sigma}$ is $\tau_{\bar{\sigma}}(i,\delta) = \delta + \sum_{0 \leq j < i} \delta_j$. The trajectory $\bar{\sigma}$ *diverges* if for all $\tau \in \mathsf{R}^+$ there is some position (i,δ) of $\bar{\sigma}$ with $\tau_{\bar{\sigma}}(i,\delta) = \tau$. If Π is a set of trajectories, we write Π^{div} for the set of divergent trajectories in Π. ■

We identify two trajectories if they can be transformed into one another by repeated replacement of a state σ with the pair $\sigma \xrightarrow{0} \sigma$. Given a trajectory $\bar{\sigma}$ and a position (i,δ) of $\bar{\sigma}$, we write $\bar{\sigma}[..(i,\delta)]$ for the finite prefix

$$\bar{\sigma}(0,0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{i-1}} \bar{\sigma}(i,0) \xrightarrow{\delta} \bar{\sigma}(i,\delta)$$

up to position (i,δ), and $\bar{\sigma}[(i,\delta)..]$ for the infinite suffix

$$\bar{\sigma}(i,\delta) \xrightarrow{\delta_i - \delta} \bar{\sigma}(i+1,0) \xrightarrow{\delta_{i+1}} \bar{\sigma}(i+2,0) \xrightarrow{\delta_{i+2}} \cdots$$

starting with position (i,δ). Note that $\bar{\sigma}[(i,\delta)..]$ is again a trajectory. By

$$s_i(\bar{\sigma}) \;=\; \{\bar{\sigma}(i,0) + \delta \mid 0 < \delta < \delta_i\}$$

we denote the set of states in the i-th segment of the trajectory $\bar{\sigma}$ (without the two endpoints $\bar{\sigma}(i,0)$ and $\bar{\sigma}(i,0)+\delta_i$). The positions of a trajectory $\bar{\sigma}$ are ordered lexicographically: the position (i,δ) precedes the position (j,ϵ), denoted by $(i,\delta) \prec (j,\epsilon)$, iff either $i < j$, or $i = j$ and $\delta < \epsilon$.

We model the timed behavior of a system by a divergent trajectory. This decision reflects several choices we make [AH92b]. First, the divergence requirement ensures the *progress* of time past any real number. Second, we choose time to be *weakly monotonic*: since a delay δ_i can be 0, time need not advance between consecutive transitions. The weak monotonicity of time allows the modeling of simultaneous activities by sequential interleaving. Not all researchers find this abstraction convenient. Our preference for weak monotonicity is no precondition for any of the results we will present and may be reversed. Third, we restrict ourselves to the modeling of systems that satisfy the condition of *finite variability*: since a trajectory is a countable sequence of states (and delays), the number of transitions in any bounded time interval of a divergent trajectory is finite.

The possible behaviors of a system are collected in a set of real-time trajectories. This set is closed under future, past, and fusion. Future and past closure result from the abstraction of initial states of the system. Fusion closure asserts that the future evolution of the system is completely determined by the present state of the system and does not depend on its past: given a fusion-closed set Π of trajectories and a state σ, the subset of trajectories in Π that start from σ form a tree structure of states with root σ.

Definition 2.3 (future, past, and fusion closure) A set Π of trajectories is *future-closed* (or *suffix-closed*) if for all trajectories $\bar{\sigma} \in \Pi$ and all positions π of $\bar{\sigma}$, the trajectory $\bar{\sigma}[\pi..]$ is in Π. The set Π is *past-closed* if for all trajectories $\bar{\sigma} \in \Pi$, all states $\sigma \in \Sigma$, and all delays $\delta \in \mathsf{R}^+$, if $\bar{\sigma}(0,0) = \sigma + \delta$ then the trajectory

$$\sigma \stackrel{\delta+\delta_0}{\rightarrow} \bar{\sigma}(1,0) \stackrel{\delta_1}{\rightarrow} \bar{\sigma}(2,0) \stackrel{\delta_2}{\rightarrow} \bar{\sigma}(3,0) \stackrel{\delta_3}{\rightarrow} \cdots$$

is in Π. A set Π of trajectories is *fusion-closed* if for all trajectories $\bar{\sigma}, \bar{\sigma}' \in \Pi$ and all positions (i,δ) of $\bar{\sigma}$ and (j,ϵ) of $\bar{\sigma}'$, if $\bar{\sigma}(i,\delta) = \bar{\sigma}'(j,\epsilon)$ then the trajectory

$$\bar{\sigma}(0,0) \stackrel{\delta_0}{\rightarrow} \cdots \stackrel{\delta_{i-1}}{\rightarrow} \bar{\sigma}(i,0) \stackrel{\delta}{\rightarrow} \bar{\sigma}(i,\delta) \stackrel{\delta'_j-\epsilon}{\rightarrow} \bar{\sigma}'(j+1,0) \stackrel{\delta'_{j+1}}{\rightarrow} \bar{\sigma}'(j+2,0) \stackrel{\delta'_{j+2}}{\rightarrow} \cdots$$

is in Π. ■

It is not difficult to check that the three closure conditions are pairwise independent. Also note that the fusion closure of a set Π of trajectories implies a form of *stutter closure*: for all trajectories $\bar{\sigma} \in \Pi$ and all positions (i,δ) of $\bar{\sigma}$, the trajectory

$$\bar{\sigma}(0,0) \stackrel{\delta_0}{\rightarrow} \cdots \stackrel{\delta_{i-1}}{\rightarrow} \bar{\sigma}(i,0) \stackrel{\delta}{\rightarrow} \bar{\sigma}(i,\delta) \stackrel{\delta_i-\delta}{\rightarrow} \bar{\sigma}(i+1,0) \stackrel{\delta_{i+1}}{\rightarrow} \bar{\sigma}(i+2,0) \stackrel{\delta_{i+2}}{\rightarrow} \cdots$$

is in Π. This condition, which is sometimes called *time additivity* [NS91], asserts that any delay of $\delta_1 + \delta_2$ time units can be split into two consecutive delays of δ_1 and δ_2 time units, respectively. We write $\Gamma(\bar{\sigma})$ for the stutter closure of a trajectory $\bar{\sigma}$; that is, $\Gamma(\bar{\sigma})$ is the smallest stutter-closed set of trajectories that contains $\bar{\sigma}$.

Definition 2.4 (premodel and real-time system) A *premodel* is a set of trajectories that is future-closed, past-closed, and fusion-closed. A *real-time system* is a premodel that contains only divergent trajectories. Given a premodel Π, a state $\sigma \in \Sigma$ is *reachable* in Π if there is some trajectory $\bar{\sigma} \in \Pi$ and some position π of $\bar{\sigma}$ such that $\bar{\sigma}(\pi) = \sigma$. We write Σ_Π for the set of states that are reachable in Π. ■

If a set Π of trajectories is closed under future (past; fusion), then so is the subset $\Pi^{div} \subseteq \Pi$ of divergent trajectories. We conclude that every premodel subsumes a real-time system.

Proposition 2.1 *If Π is a premodel, then Π^{div} is a real-time system.*

2.2 Safety and divergence safety

A safety property is a set of admissible system behaviors that is closed under certain limit behaviors: a set Π of infinite state sequences is closed under limits iff for any infinite state sequence $\bar{\sigma}$, whenever all finite prefixes of $\bar{\sigma}$ are prefixes of sequences in Π, then the limit sequence $\bar{\sigma}$ itself is in Π [ADS86]. We define the corresponding notion of safety for real-time trajectories.

Definition 2.5 (safety) A set Π of trajectories is *safe* if for any trajectory $\bar{\sigma}$, if for all positions π of $\bar{\sigma}$ there is some trajectory $\bar{\sigma}' \in \Pi$ and some position π' of $\bar{\sigma}'$ such that $\bar{\sigma}[..\pi] = \bar{\sigma}'[..\pi']$, then $\bar{\sigma} \in \Pi$. The *safety closure* $\overline{\Pi}$ is the smallest safe set of trajectories that subsumes Π. ■

Suppose that a real-time system Π contains a trajectory whose initial state is σ. Since Π is closed under stuttering, its safety closure $\overline{\Pi}$ must contain the trajectory

$$\sigma \xrightarrow{0} \sigma \xrightarrow{0} \sigma \xrightarrow{0} \cdots,$$

which does not diverge. It follows that no nonempty real-time system is safe. Thus we define the less stringent requirement of divergence safety: only the divergent limit trajectories must be present.[1]

Definition 2.6 (divergence safety) A set Π of divergent trajectories is *divergence-safe* if for any divergent trajectory $\bar{\sigma}$, if for all positions π of $\bar{\sigma}$ there is some trajectory $\bar{\sigma}' \in \Pi$ and some position π' of $\bar{\sigma}'$ such that $\bar{\sigma}[..\pi] = \bar{\sigma}'[..\pi']$, then $\bar{\sigma} \in \Pi$. The *divergence-safety closure* $\widetilde{\Pi}$ is the smallest divergence-safe set of trajectories that subsumes Π. ■

Example 2.1 Let Π_1 be the real-time system that changes the value of a proposition p from *false* to *true* before time 10; that is, Π_1 is the set of all divergent trajectories $\bar{\sigma}$ with some position π and some clock $x \in C$ such that for all positions π' of $\bar{\sigma}$,

1. if $\pi' \prec \pi$ then $\bar{\sigma}(\pi')(p) = \mathit{false}$ and $0 \leq \bar{\sigma}(\pi')(x) < 10$,
2. if $\pi' \succeq \pi$ then $\bar{\sigma}(\pi')(p) = \mathit{true}$, and
3. if $\bar{\sigma}(\pi')(x) = 0$ then $\tau_{\bar{\sigma}}(\pi') = 0$.

It is not difficult to check that the set Π_1 is indeed closed under future, past, and fusion.

The real-time system Π_1 is divergence-safe. To see this, take a divergent trajectory $\bar{\sigma}$ such that every finite prefix of $\bar{\sigma}$ is a prefix of a trajectory in Π_1. Since $\bar{\sigma}$ diverges, it has a position π with $\tau_{\bar{\sigma}}(\pi) \geq 10$. Since $\bar{\sigma}[..\pi]$ is a prefix of a trajectory in Π_1, it satisfies all three conditions for some position $\pi' \preceq \pi$ and some clock x. Furthermore, since $\bar{\sigma}[..\pi']$ is a prefix of some trajectory in Π_1 for all positions $\pi' \succ \pi$ of $\bar{\sigma}$, also $\bar{\sigma}(\pi')(p) = \mathit{true}$ and $\bar{\sigma}(\pi')(x) > 0$ for all $\pi' \succ \pi$.

Now consider the real-time system Π_2 that changes the value of p eventually; that is, Π_2 is the set of all divergent trajectories $\bar{\sigma}$ with some position π such that for all positions π' of $\bar{\sigma}$,

1. if $\pi' \prec \pi$ then $\bar{\sigma}(\pi')(p) = \mathit{false}$, and
2. if $\pi' \succeq \pi$ then $\bar{\sigma}(\pi')(p) = \mathit{true}$.

The real-time system Π_2 is not divergence-safe. To see this, let $\bar{\sigma}$ be a divergent trajectory such that $\bar{\sigma}(\pi) = \mathit{false}$ for all positions π of $\bar{\sigma}$. Then every finite prefix of $\bar{\sigma}$ is the prefix of a trajectory in Π_2, but $\bar{\sigma}$ itself is not in Π_2. ■

[1] The reader familiar with [Hen92] will notice that the concept of divergence safety corresponds to the notion of *safety relative to the divergence of time*.

To close a set Π of trajectories under (divergent) limits, we add a (divergent) trajectory $\bar{\sigma}$ if all finite prefixes of $\bar{\sigma}$ are prefixes of trajectories in Π. This process need not be iterated. It follows that the divergence-safety closure of a real-time system Π can be obtained from the safety closure of Π simply by dropping all trajectories that do not diverge.

Proposition 2.2 *For all sets* Π *of divergent trajectories,* $\widetilde{\Pi} = \overline{\Pi}^{div}$*.*

Corollary 2.1 *If the premodel* Π *is safe, then the real-time system* Π^{div} *is divergence-safe.*

2.3 Next-state relations

Safe premodels and divergence-safe real-time systems can be defined by a binary transition relation on the state space, which indicates for each state $\sigma \in \Sigma$ the possible transition successors of σ. Each transition successor $\sigma' \in \Sigma$ of σ results from σ' by modifying the propositional component of the state and by resetting some of the clocks.

Definition 2.7 (transition relation) For a binary relation $S \subseteq \Sigma^2$ on the states, we write Σ_S for the union of the domain and the range of S. The relation S is *past-closed* if for all states $\sigma \in \Sigma$ and all delays $\delta \in \mathsf{R}^+$, if $\sigma + \delta \in \Sigma_S$ then $\sigma \in \Sigma_S$. The relation S is *self-reflexive* if for all $\sigma \in \Sigma_S$, $(\sigma, \sigma) \in S$. A *transition relation* $S \subseteq \Sigma^2$ is a past-closed self-reflexive binary relation on the states such that $(\sigma, \sigma') \in S$ implies for all clocks $x \in C$, either $\sigma'(x) = \sigma(x)$ or $\sigma'(x) = 0$. ■

Every transition relation S determines a two-phase next-state relation $\triangleright_S$ that first advances time, by any amount, and then performs a transition. The iteration of the two-phase relation generates real-time trajectories. Transition relations are required to be self-reflexive so that infinite trajectories can be generated without the possibility of deadlock.

Definition 2.8 (generated trajectory) Given a transition relation S and a delay $\delta \in \mathsf{R}^+$, let $\rightarrow^\delta = \{(\sigma, \sigma + \delta) \mid \sigma \in \Sigma\}$ and let $\triangleright_S^\delta = \rightarrow^\delta \circ S$. The transition relation S defines the *next relation* $\triangleright_S = \bigcup_{\delta \in \mathsf{R}^+} \triangleright_S^\delta$. A trajectory

$$\bar{\sigma} = \sigma_0 \xrightarrow{\delta_0} \sigma_1 \xrightarrow{\delta_1} \sigma_2 \xrightarrow{\delta_2} \sigma_3 \xrightarrow{\delta_3} \cdots$$

is *generated* by the transition relation S if $\sigma_i \triangleright_S^{\delta_i} \sigma_{i+1}$ for all $i \geq 0$. We say that $\bar{\sigma}$ is an S*-trajectory* and write Π_S for the set of S-trajectories. ■

It is not difficult to check that the set Π_S of trajectories that are generated by a transition relation S is closed under future, past, fusion, and limits.

Proposition 2.3 *For any transition relation* S*, the set* Π_S *of* S*-trajectories is a safe premodel.*

Corollary 2.2 *For any transition relation* S*, the set* Π_S^{div} *of divergent* S*-trajectories is a divergence-safe real-time system.*

Every transition relation S defines, then, a unique premodel, Π_S, which is safe, and a unique real-time system, Π_S^{div}, which is divergence-safe. Indeed, we show that the four closure conditions (closure under future, past, fusion, and limits) are both necessary and sufficient for a set of trajectories to be definable by a transition relation.

Definition 2.9 (induced relation) A set Π of trajectories *induces* the smallest relation $S_\Pi \subseteq \Sigma^2$ such that for all trajectories

$$\sigma_0 \xrightarrow{\delta_0} \sigma_1 \xrightarrow{\delta_1} \sigma_2 \xrightarrow{\delta_2} \sigma_3 \xrightarrow{\delta_3} \cdots$$

in Π and for all nonnegative integers i, $(\sigma_i + \delta_i, \sigma_{i+1}) \in S_\Pi$. ■

Let Π be a set of trajectories. Since the induced relation S_Π is defined locally on consecutive states of trajectories in Π, if two sets of trajectories have the same safety closure, then they induce the same transition relation.

Proposition 2.4 *For all sets* Π *of trajectories,* $S_\Pi = S_{\overline{\Pi}}$.

Corollary 2.3 *For all sets* Π *of divergent trajectories,* $S_\Pi = S_{\widetilde{\Pi}}$.

If a set Π of trajectories is future-closed, past-closed, and stutter-closed, then the induced relation S_Π is past-closed and self-reflexive (namely, reflexive on the states Σ_Π that are reachable in Π).

Proposition 2.5 *If* Π *is a premodel, then the induced relation* S_Π *is a transition relation.*

We now generalize a well-known result for infinite state sequences: a set of infinite state sequences is generated by a binary relation on the states iff it is closed under suffixes, fusion, and limits [Eme83] (closure under stuttering corresponds to the requirement of self-reflexivity for the generating relation). Every safe premodel Π is definable by a transition relation, namely S_Π, which generates precisely the trajectories in Π. Likewise, every divergence-safe real-time system Π can be defined by the transition relation S_Π.

Proposition 2.6 *For all premodels* Π, $\Pi_{S_\Pi} = \overline{\Pi}$.

Proof. First observe that $\overline{\Pi} \subseteq \Pi_{S_{\overline{\Pi}}}$ and thus, by Proposition 2.4, also $\overline{\Pi} \subseteq \Pi_{S_\Pi}$.

To see that $\Pi_{S_\Pi} \subseteq \overline{\Pi}$, consider a trajectory $\bar\sigma$ such that for all $i \geq 0$, there is a trajectory $\bar\sigma' \in \Pi$ and a nonnegative integer j with $\bar\sigma_i + \delta_i = \bar\sigma'_j + \delta'_j$ and $\bar\sigma_{i+1} = \bar\sigma'_{j+1}$. Since Π is future-closed, for all $i \geq 0$ there is a trajectory $\bar\sigma'' \in \Pi$ with $\bar\sigma_i + \delta_i = \bar\sigma''_0$ and $\delta''_0 = 0$ and $\bar\sigma_{i+1} = \bar\sigma''_1$. Since Π is past-closed, for all $i \geq 0$ there is a trajectory $\bar\sigma''' \in \Pi$ with $\bar\sigma_i = \bar\sigma'''_0$ and $\delta_i = \delta'''_0$ and $\bar\sigma_{i+1} = \bar\sigma'''_1$. From repeated application of the fusion closure of Π, it follows that for all positions π of $\bar\sigma$ there is a trajectory $\bar\sigma^+ \in \Pi$ and a position π^+ of $\bar\sigma^+$ with $\bar\sigma[..\pi] = \bar\sigma^+[..\pi^+]$. Since $\overline{\Pi}$ is closed under limits, it follows that $\bar\sigma \in \overline{\Pi}$. ■

Corollary 2.4 *For all real-time systems* Π, $\Pi^{div}_{S_\Pi} = \widetilde{\Pi}$.

2.4 Nonzenoness

For untimed systems (premodels), there is a pleasing one-to-one correspondence between safety properties (safe premodels) and next-state (transition) relations. In particular, every safe premodel Π can be defined operationally, by the induced transition relation S_Π, and S_Π is the only transition relation that generates Π.

Proposition 2.7 *For all transition relations* S, $S_{\Pi_S} = S$.

Corollary 2.5 *For all premodels* Π *and transition relations* S*,* $\Pi = \Pi_S$ *iff both* Π *is safe and* $S = S_\Pi$*.*

Corollary 2.4 allows us to define divergence-safe real-time systems operationally, by their transition relations, just as untimed safety properties can be defined by next-state relations. However, while there is a unique transition relation that generates a safe premodel Π_1 (namely S_{Π_1}), there are many transition relations that generate the same divergence-safe real-time system Π_2. We call a transition relation S that generates Π_2 nonzeno iff it suggests an operational semantics for Π_2; that is, iff every finite prefix of an S-trajectory can be extended to a divergent S-trajectory [AL92, Hen92].[2] We will see that only one of the transition relations that generate Π_2 is nonzeno (namely S_{Π_2}), and thus a suitable operational definition for the divergence-safe real-time system Π_2.

Definition 2.10 (nonzenoness) A transition relation S is *nonzeno* if for all S-trajectories $\bar{\sigma}$ and all positions π of $\bar{\sigma}$, there is a divergent S-trajectory $\bar{\sigma}'$ with $\bar{\sigma}'(0,0) = \bar{\sigma}(\pi)$. ■

A nonzeno transition relation S suggests a notion of execution for the divergence-safe real-time system Π_S^{div} defined by S. Each execution step consists of two phases, namely, a time delay followed by a transition from S. A simple-minded interpreter that iterates execution steps will generate precisely all prefixes of the trajectories in Π_S^{div}. If, on the other hand, the transition relation S is zeno, then the simple-minded interpreter can paint itself into a corner and arrive at a state from which it cannot let time diverge (thus generating prefixes of sequences in Π_S that are not prefixes of sequences in Π_S^{div}).

Example 2.2 The divergence-safe real-time system Π_1 of Example 2.1 induces the transition relation

$$S_{\Pi_1} = \left\{ (\sigma,\sigma') \mid \begin{array}{l} (\sigma(p) = \mathit{false} \wedge \sigma(x) < 10 \wedge \sigma'(p) = \mathit{true} \wedge \sigma'(x) = \sigma(x)) \vee \\ (\sigma(p) = \mathit{false} \wedge \sigma(x) < 10 \wedge \sigma' = \sigma) \vee \\ (\sigma(p) = \mathit{true} \wedge \sigma' = \sigma) \end{array} \right\}.$$

The transition relation S_{Π_1} is nonzeno and Π_1 is the set of all divergent S_{Π_1}-trajectories. Now consider the transition relation

$$S_2 = S_{\Pi_1} \cup \{(\sigma,\sigma') \mid \sigma(x) < 20 \wedge \sigma' = \sigma\}.$$

While the set of all divergent S_2-trajectories is still Π_1, the transition relation S_2 is zeno. To see this, observe that no nontrivial finite prefix of the S_2-trajectory

$$(\mathit{false}, 0) \xrightarrow{15} (\mathit{false}, 15) \xrightarrow{0} (\mathit{false}, 15) \xrightarrow{0} (\mathit{false}, 15) \xrightarrow{0} \cdots$$

can be extended to a divergent S_2-trajectory (here $(\mathit{false}, 0)$ stands for the state σ with $\sigma(p) = \mathit{false}$ and $\sigma(x) = 0$). Although both transition relations S_{Π_1} and S_2 define the same divergence-safe real-time system, namely Π_1, only the former relation suggests an operational semantics for Π_1: iterate time delays with transitions from S_{Π_1}. ■

[2] The reader familiar with the concept of machine closure [AL88] will realize that the nonzenoness requirement is a machine closure condition, namely, machine closure with respect to the liveness property that asserts the divergence of time.

We now show the real-time analogues of Proposition 2.7 and Corollary 2.5, which reveal a one-to-one correspondence between divergence-safe real-time systems and nonzeno transition relations. It follows that every divergence-safe real-time system Π can be defined operationally, by the induced transition relation S_Π, and S_Π is the only nonzeno transition relation that defines Π.

Proposition 2.8 *A transition relation S is nonzeno iff $S_{\Pi_S^{div}} = S$.*

Proof. Let S be a transition relation. First observe that $S_{\Pi_S^{div}} \subseteq S$.

Now suppose that S is nonzeno. To see that $S \subseteq S_{\Pi_S^{div}}$, consider a pair $(\sigma, \sigma') \in S$. Since S is self-reflexive, there is an S-trajectory that starts with the prefix $\sigma \xrightarrow{0} \sigma'$. Since S is nonzeno, there is a divergent S-trajectory that starts with the prefix $\sigma \xrightarrow{0} \sigma'$. It follows that $(\sigma, \sigma') \in S_{\Pi_S^{div}}$.

Finally, we suppose that S is zeno and show that $S \not\subseteq S_{\Pi_S^{div}}$. Since S is zeno, there is a state σ that occurs on an S-trajectory but does not occur on any divergent S-trajectory. Then $(\sigma, \sigma) \notin S_{\Pi_S^{div}}$, and $(\sigma, \sigma) \in S$ because S is self-reflexive. ■

Corollary 2.6 *For all real-time systems Π and nonzeno transition relations S, $\Pi = \Pi_S^{div}$ iff both Π is divergence-safe and $S = S_\Pi$.*

3 Real-time Programs

We present two languages for defining divergence-safe real-time systems—a guarded-command programming language with clocks and finite-state machines with clocks. Both languages specify transition relations.

3.1 State predicates

To assert properties of individual states, we first define a language of state predicates. Recall that each state in Σ is an interpretation of the boolean propositions in P and the real-valued clocks in C.

Definition 3.1 (state predicate) The set of *state predicates* is defined inductively by the grammar

$$\phi \ ::= \ p \mid x \leq d \mid c \leq y \mid x + c \leq y + d \mid \neg\phi \mid \phi_1 \vee \phi_2$$

for propositions $p \in P$, clocks $x, y \in C$, and nonnegative integer constants $c, d \in \mathbb{N}$. ■

State predicates are interpreted over states. Given a state $\sigma \in \Sigma$, a state predicate ϕ evaluates to a truth value $\sigma(\phi) \in \{\mathit{true}, \mathit{false}\}$ in the standard way. The state σ *satisfies* ϕ, denoted by $\sigma \models \phi$, if $\sigma(\phi) = \mathit{true}$. We write $[\![\phi]\!] \subseteq \Sigma$ for the set of those states that satisfy the state predicate ϕ, and we call them ϕ-states. Two state predicates ϕ_1 and ϕ_2 are *equivalent* if $[\![\phi_1]\!] = [\![\phi_2]\!]$.

Standard abbreviations for state predicates, such as true, $p \rightarrow x = 2$, and $2 < y < 5$, are defined as usual. For state predicates ϕ and ϕ' and a proposition $p \in P$, we write $\phi[p := \phi']$ for the state predicate that results from ϕ by replacing each occurrence of p with ϕ'; similarly, for clocks $x, y \in C$ and a constant $c \in \mathbb{N}$, by $\phi[x := c]$ (and $\phi[x := y + c]$) we denote the state predicate that is obtained from ϕ by replacing each occurrence of x with c (or $y + c$, respectively). Given a set $A = \{v_1 := a_1, \ldots, v_n := a_n\}$ of replacements, we write $\phi[A]$ for the state predicate that results from ϕ by simultaneously replacing each variable v_i with the corresponding expression a_i. Finally, for a delay $\delta \in \mathsf{R}^+$ and the set $A = \{x := x + \delta \mid x \in C\}$ of replacements, we write $\phi + \delta$ for the state predicate $\phi[A]$.

The satisfiability problem for state predicates is obviously no simpler than boolean satisfiability. Indeed, both satisfiability problems are equally difficult.

Theorem 3.1 *The satisfiability problem for state predicates is NP-complete.*

Proof. Given a state predicate ϕ, let k be the largest constant in ϕ and let n be the number of clocks in ϕ. Suppose that there is a satisfying interpretation σ for ϕ and $\sigma(x_1) \leq \cdots \leq \sigma(x_n)$ for the clocks $x_1, \ldots, x_n$ of ϕ. Construct an interpretation σ' that agrees with σ on all propositions and the ordering of clock values such that

$$\sigma'(x_i) - \sigma'(x_{i-1}) \;=\; \begin{cases} \sigma(x_i) - \sigma(x_{i-1}) & \text{if } \sigma(x_i) - \sigma(x_{i-1}) \leq k, \\ k+1 & \text{otherwise} \end{cases}$$

(assuming that $\sigma(x_0) = \sigma'(x_0) = 0$). Then σ' satisfies ϕ as well. Since no clock is assigned a value greater than $(k+1) \cdot n$, the interpretation σ' can be guessed and checked in time polynomial in the length of ϕ. ■

3.2 Evolving state predicates

State predicates define static sets of states. If time advances from a state, the clock values change, and so may the value of a state predicate. We describe the motion of state sets in time using the binary *evolves-to* operator $\rightsquigarrow$. Consider two state predicates ϕ_1 and ϕ_2. The evolving state predicate $\phi_1 \rightsquigarrow \phi_2$ defines the set of states from which a state satisfying ϕ_2 can be reached through advancing time by some nonnegative amount, and ϕ_1 is satisfied until ϕ_2 is reached.

Definition 3.2 (evolving state predicate) The set of *evolving state predicates* is obtained by adding the evolves-to operator $\rightsquigarrow$ to the grammar of state predicates: if ϕ_1 and ϕ_2 are evolving state predicates, then so is $\phi_1 \rightsquigarrow \phi_2$. A state $\sigma \in \Sigma$ *satisfies* the evolving state predicate $\phi_1 \rightsquigarrow \phi_2$, written $\sigma \models \phi_1 \rightsquigarrow \phi_2$, if there is a nonnegative real δ such that $\sigma + \delta \models \phi_2$ and for all nonnegative reals $\epsilon \leq \delta$, $\sigma + \epsilon \models \phi_1$. ■

It is perhaps most intuitive to view the evolves-to operator geometrically. Suppose that the state predicate ϕ contains n clock variables and no propositions. Then ϕ defines a finite union of polyhedra in n-dimensional space and the evolving state predicate $true \rightsquigarrow \phi$ defines the shadow of ϕ for a light source at ∞^n:

$$[\![true \rightsquigarrow \phi]\!] \;=\; [\![\exists \delta \geq 0.\, \phi + \delta]\!].$$

Example 3.1 Consider the two-dimensional state predicates

$$\begin{aligned} \phi_1 &= (0 \leq x \leq 2 \,\wedge\, y \geq 1), \\ \phi_2 &= (1 \leq x \leq 3 \,\wedge\, 2 \leq y \leq 4). \end{aligned}$$

The evolving state predicates $true \rightsquigarrow \phi_2$ and $\phi_1 \rightsquigarrow \phi_2$ define, then, the two polygons

$$\begin{aligned} [\![true \rightsquigarrow \phi_2]\!] &= [\![0 \leq x \leq 3 \,\wedge\, 0 \leq y \leq 4 \,\wedge\, x - 1 \leq y \leq x + 3]\!], \\ [\![\phi_1 \rightsquigarrow \phi_2]\!] &= [\![0 \leq x \leq 2 \,\wedge\, 1 \leq y \leq 4 \,\wedge\, x \leq y \leq x + 3]\!] \end{aligned}$$

shown in Figure 1. Note that both evolving state predicates are equivalent to pure state predicates. We shall see that this is no accident. ■

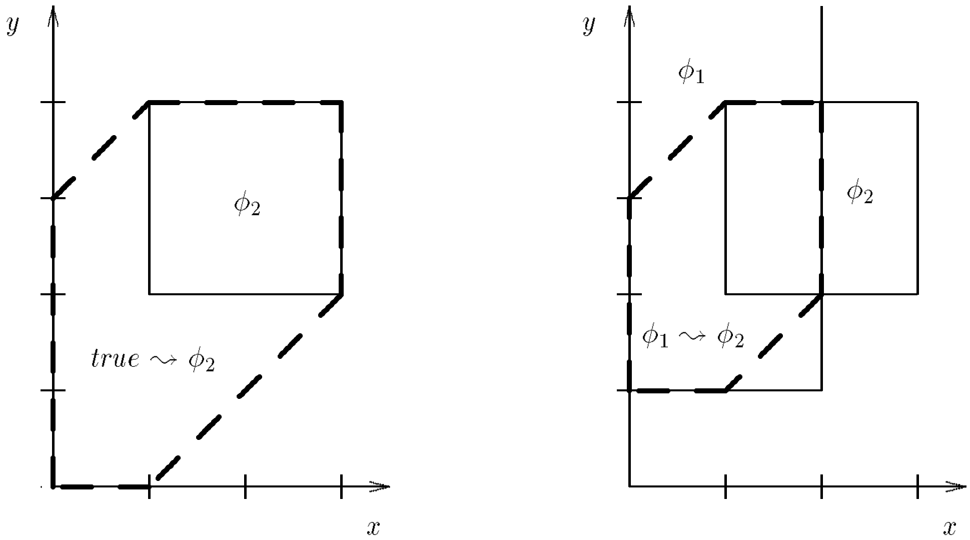

Figure 1: The graphic representations of $true \rightsquigarrow \phi_2$ and $\phi_1 \rightsquigarrow \phi_2$

Viewed logically, the evolves-to operator extends the language of state predicates with a restricted form of quantification over the real numbers R: the evolving state predicate $\phi_1 \rightsquigarrow \phi_2$ is equivalent to the quantified formula

$$\exists\delta \in \mathsf{R}.\,(\delta \geq 0 \,\wedge\, \phi_2 + \delta \,\wedge\, \forall\epsilon \in \mathsf{R}.\,(0 \leq \epsilon \leq \delta \,\rightarrow\, \phi_1 + \epsilon)).$$

By eliminating both quantifiers, we show that evolving state predicates are no more expressive than state predicates. Since Tarski showed that the theory of the reals with order and addition admits quantifier elimination, this should come as no surprise. To ensure that the quantifier-free formula obtained by eliminating the evolves-to operator is a state predicate, however, we cannot use the near-optimal decision procedure of Ferrante and Rackoff for the first-order theory of real addition with order [FR75]. We provide a nonelementary translation from evolving state predicates to state predicates; the exact complexity of the satisfiability problem for evolving state predicates is an open problem.

Theorem 3.2 *For any evolving state predicate ϕ we can construct an equivalent state predicate $|\phi|$. Moreover, the integer constants occurring in $|\phi|$ are bounded by the largest integer constant in ϕ.*

Proof. We repeat eliminating the innermost evolves-to operator. Consider the subformula $\phi_1 \rightsquigarrow \phi_2$ for state predicates ϕ_1 and ϕ_2:

$$[\![\phi_1 \rightsquigarrow \phi_2]\!] \;=\; [\![\exists\delta \in \mathsf{R}.\,(\delta \geq 0 \,\wedge\, \phi_2 + \delta \,\wedge\, \neg\exists\epsilon \in \mathsf{R}.\,(0 \leq \epsilon \leq \delta \,\wedge\, \neg\phi_1 + \epsilon))]\!].$$

We proceed in two steps that eliminate the two quantifiers. First we eliminate the inner existential quantifier. All atoms in the scope ϕ' of the quantifier $\exists\epsilon.\,\phi'$ are of the forms $0 \leq \epsilon$, $\epsilon \leq \delta$, p, $x + \epsilon \leq d$, $c \leq y + \epsilon$, and $x + c \leq y + d$. We convert ϕ' into disjunctive normal form, which may cause an exponential blowup. Observe that the existential quantifier distributes over disjunction and atoms of the forms p and $x + c \leq y + d$. We rearrange all remaining inequalities so that they provide lower and upper bounds on ϵ, and we then solve each disjunct for ϵ, which may cause a quadratic blowup:

$$\begin{array}{l}[\![\exists\epsilon.\,(0 \leq \epsilon \;\wedge\; \epsilon \leq \delta \;\wedge\; \bigwedge_{i\in I} \epsilon \leq d_i - x_i \;\wedge\; \bigwedge_{j\in J} c_j - y_j \leq \epsilon)]\!] \;= \\ [\![0 \leq \delta \;\wedge\; \bigwedge_{i\in I} 0 \leq d_i - x_i \;\wedge\; \bigwedge_{j\in J} c_j - y_j \leq \delta \;\wedge\; \bigwedge_{i\in I, j\in J} c_j - y_j \leq d_i - x_i]\!] \;= \\ [\![0 \leq \delta \;\wedge\; \bigwedge_{i\in I} x_i \leq d_i \;\wedge\; \bigwedge_{j\in J} c_j \leq y_j + \delta \;\wedge\; \bigwedge_{i\in I, j\in J} x_i + c_j \leq y_j + d_i]\!].\end{array}$$

Next we eliminate the outer existential quantifier. All atoms in the scope ϕ'' of the quantifier $\exists\delta.\,\phi''$ are of the forms $0 \leq \delta$, p, $x + \delta \leq d$, $c \leq y + \delta$, $x + c \leq y + d$, and $x \leq d$. We convert ϕ'' into disjunctive normal form and solve each disjunct for δ:

$$\begin{array}{l}[\![\exists\delta.\,(0 \leq \delta \;\wedge\; \bigwedge_{i\in I} \delta \leq d_i - x_i \;\wedge\; \bigwedge_{j\in J} c_j - y_j \leq \delta)]\!] \;= \\ [\![\bigwedge_{i\in I} 0 \leq d_i - x_i \;\wedge\; \bigwedge_{i\in I, j\in J} c_j - y_j \leq d_i - x_i]\!] \;= \\ [\![\bigwedge_{i\in I} x_i \leq d_i \;\wedge\; \bigwedge_{i\in I, j\in J} x_i + c_j \leq y_j + d_i]\!].\ \blacksquare\end{array}$$

3.3 Guarded-command real-time programs

A real-time program consists of a set of guarded commands and a program invariant. Each transition corresponds to the execution of a guarded command. The guard of a command refers to the values of propositions and clocks; it asserts a *necessary* condition for the corresponding transition to take place. The program invariant also refers to the values of propositions and clocks; it must not be violated by letting time advance, and thus asserts a *sufficient* condition for a transition to take place. For example, the guard $x \geq 5$, for a clock x, puts a lower bound on the time of the corresponding transition, which cannot take place before the clock x reaches the value 5. The invariant $x < 8$, on the other hand, implies an upper bound on the time of the next transition: some transition must take place before the clock x reaches the value 8. When a transition is taken, some of the propositions are assigned new values and some of the clocks are reset to 0.

Definition 3.3 (real-time program: syntax) A (*guarded-command*) *real-time program* $\mathcal{P} = (G, \phi^{\Box})$ consists of

1. G—the *program body*, a set of guarded commands. Each *guarded command* $g \in G$ is of the form $\psi \to A$, for a state predicate ψ (the *guard* of the command) and a set $A = \{v := a_v \mid v \in P \cup C\}$ of simultaneous assignments such that for all propositions $p \in P$, the expression a_p is a state predicate, and for all clocks $x \in C$, the expression a_x is either 0 (i.e., the clock x is reset) or x (i.e., the clock x is left unchanged).
2. $\phi^{\Box}$—the *program invariant*, a state predicate. We require $\phi^{\Box}$ to be *past-closed*; that is, for all states $\sigma \in \Sigma$ and all delays $\delta \in \mathsf{R}^+$, if $\sigma + \delta \models \phi^{\Box}$ then $\sigma \models \phi^{\Box}$.

$\blacksquare$

A guarded command defines a partial function from Σ to Σ. Let $g = \psi \to A$ be the guarded command with $A = \{v := a_v \mid v \in P \cup C\}$, and let $\sigma \in \Sigma$ be a state. The guarded command g is *enabled* in the state σ if $\sigma \models \psi$. Any guarded command that is enabled in σ may be executed in σ. The execution of g, in particular, leads to the state $\sigma[A]$, where $\sigma[A](v) = \sigma(a_v)$ for all variables $v \in P \cup C$.

Definition 3.4 (real-time program: semantics) The real-time program $\mathcal{P} = (G, \phi^{\Box})$ defines the transition relation $S_{\mathcal{P}}$ such that $(\sigma, \sigma') \in S_{\mathcal{P}}$ if

1. either $\sigma' = \sigma$ or for some guarded command $g = \psi \rightarrow A$ in G, $\sigma \models \psi$ and $\sigma' = \sigma[A]$,
2. $\sigma \models \phi^{\Box}$ and $\sigma' \models \phi^{\Box}$.

Each divergent $S_{\mathcal{P}}$-trajectory is called a *run* of $\mathcal{P}$. We write $\Pi_{\mathcal{P}} = \Pi^{div}_{S_{\mathcal{P}}}$ for the set of runs of $\mathcal{P}$, and $\Sigma_{\mathcal{P}}$ for the set of states that are reachable in $\Pi_{\mathcal{P}}$. Two real-time programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *equivalent* if they have the same runs (i.e., $\Pi_{\mathcal{P}_1} = \Pi_{\mathcal{P}_2}$). The real-time program $\mathcal{P}$ is *nonzeno* if the transition relation $S_{\mathcal{P}}$ is nonzeno. ■

A real-time program $\mathcal{P} = (G, \phi^{\Box})$ defines, then, the safe premodel $\Pi_{S_{\mathcal{P}}}$ and the divergence-safe real-time system $\Pi_{\mathcal{P}}$. Each transition of $\Pi_{\mathcal{P}}$ corresponds to the execution of a guarded command in G. The past closure of the invariant $\phi^{\Box}$ ensures that $\sigma \models \phi^{\Box}$ for all states $\sigma \in \Sigma_{\mathcal{P}}$ that occur on some run of $\mathcal{P}$.

Example 3.2 The divergence-safe real-time system Π_1 of Example 2.1 is defined by the real-time program $\mathcal{P}_1$ with the single guarded command $p := \mathit{true}$ and the invariant $p \vee x < 10$. (When writing guarded commands, we suppress the guard *true* and assignments of the form $v := v$.) This is because $S_{\mathcal{P}_1} = S_{\Pi_1}$.

Alternatively, the divergence-safe real-time system Π_1 can be defined by the real-time program $\mathcal{P}_2$ with the single guarded command $x < 10 \rightarrow p := \mathit{true}$ and the invariant $p \vee x < 20$. Then $\Pi_{\mathcal{P}_1} = \Pi_{\mathcal{P}_2}$, but $S_{\mathcal{P}_1} \subset S_{\mathcal{P}_2}$. While both real-time programs $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same runs, only $\mathcal{P}_1$ is nonzeno. Indeed, the transition relation defined by $\mathcal{P}_2$ is the zeno transition relation S_2 of Example 2.2. ■

The invariant $\phi^{\Box}$ of a real-time program $\mathcal{P}$ defines precisely the set $\Sigma_{S_{\mathcal{P}}}$ of states that occur on some $S_{\mathcal{P}}$-trajectory. We now show that the program $\mathcal{P}$ is nonzeno iff it has the strongest possible invariant, namely, an invariant that defines precisely the set $\Sigma_{\mathcal{P}}$ of states that occur on some run of $\mathcal{P}$.

Proposition 3.1 *Let $\mathcal{P}$ be a real-time program with the invariant $\phi^{\Box}$. Then $[\![\phi^{\Box}]\!] = \Sigma_{S_{\mathcal{P}}}$. Moreover, the real-time program $\mathcal{P}$ is nonzeno iff $[\![\phi^{\Box}]\!] = \Sigma_{\mathcal{P}}$.*

Proof. Since all states that occur on an $S_{\mathcal{P}}$-trajectory satisfy the invariant, $\Sigma_{\mathcal{P}} \subseteq \Sigma_{S_{\mathcal{P}}} \subseteq [\![\phi^{\Box}]\!]$. Now suppose that the state σ satisfies the invariant $\phi^{\Box}$. Since

$$\sigma \xrightarrow{0} \sigma \xrightarrow{0} \sigma \xrightarrow{0} \cdots$$

is an $S_{\mathcal{P}}$-trajectory, $\sigma \in \Sigma_{S_{\mathcal{P}}}$. If the real-time program $\mathcal{P}$ is nonzeno, then every state that occurs on an $S_{\mathcal{P}}$-trajectory occurs also on a run of $\mathcal{P}$ and, therefore, $\sigma \in \Sigma_{\mathcal{P}}$.

Conversely, suppose that every state that satisfies the invariant $\phi^{\Box}$ occurs on a run of $\mathcal{P}$. To show that the real-time program $\mathcal{P}$ is nonzeno, it suffices to show that any state that occurs on an $S_{\mathcal{P}}$-trajectory occurs also on a run of $\mathcal{P}$. This follows from the assumption, because all states that occur on an $S_{\mathcal{P}}$-trajectory satisfy the invariant $\phi^{\Box}$. ■

It follows that nonzeno real-time programs can be executed in a stepwise fashion: start with any state that satisfies the invariant and then, repeatedly, either choose a guarded command that is enabled and whose execution does not violate the invariant, or advance time without violating the invariant. The iteration of the next relation $\triangleright_{S_{\mathcal{P}}}$ defined by a zeno real-time program, on the other hand, may lead to a state from which time cannot diverge. In other words, the nonzeno real-time programs are precisely the real-time programs that can be executed without looking ahead. If a real-time program is zeno, then its invariant is too weak.

In Section 6, we will show that for every real-time program $\mathcal{P}$, the set $\Sigma_{\mathcal{P}}$ of states that occur on some run of $\mathcal{P}$ can be defined by a state predicate ϕ. Hence there is a nonzeno real-time program that is equivalent to $\mathcal{P}$ (replace the invariant of $\mathcal{P}$ with ϕ). Indeed, we will give an algorithm that *automatically* converts any given real-time program into an equivalent nonzeno program by strengthening the invariant. Here, let us consider a second example.

Example 3.3 Consider the real-time program $\mathcal{P}_1 = (G_1, \phi_1^{\Box})$, where α is a ternary variable (short for two boolean variables) that ranges over $\{a, b, c\}$, and x is a clock:

$$\begin{array}{rcl} G_1 & = \{ & \alpha = a \rightarrow \alpha := b, \\ & & \alpha = b \rightarrow \alpha := a, \\ & & (\alpha = b \wedge x \geq 1) \rightarrow \alpha := c \}; \\ \phi_1^{\Box} & = & (\alpha = a \wedge x \leq 4) \vee (\alpha = b \wedge x \leq 4) \vee (\alpha = c). \end{array}$$

If this program is started with $\alpha = a$ and $x = 0$, then the value of α alternates between a and b arbitrarily (but finitely) often, until α is assigned c at some point between time 1 and 4. The lower bound of 1 is imposed by the guard of the third guarded command; the upper bound of 4 is imposed by the invariant. Once α is set to c, the program does nothing but let time pass.

Now consider the real-time program $\mathcal{P}_2 = (G_2, \phi_2^{\Box})$:

$$\begin{array}{rcl} G_2 & = \{ & \alpha = a \rightarrow \alpha := b, \\ & & \alpha = b \rightarrow \alpha := a, \\ & & (\alpha = b \wedge 1 \leq x \leq 4) \rightarrow \alpha := c \}; \\ \phi_2^{\Box} & = & (\alpha = a \wedge x < 5) \vee (\alpha = b \wedge x < 5) \vee (\alpha = c). \end{array}$$

It is not difficult to check that the real-time programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent. However, while $\mathcal{P}_1$ is nonzeno, $\mathcal{P}_2$ is zeno, because the invariant $\phi_2^{\Box}$ is too weak. This is because any state σ with $\sigma(\alpha) \in \{a, b\}$ and $4 < \sigma(x) < 5$ occurs on the $S_{\mathcal{P}_2}$-trajectory

$$\sigma \xrightarrow{0} \sigma \xrightarrow{0} \sigma \xrightarrow{0} \cdots,$$

but σ does not occur on a run of $\mathcal{P}_2$. When executing $\mathcal{P}_2$, a stepwise interpreter may decide to let time pass beyond 4 in a state with $\alpha = a$ or $\alpha = b$, in which case it commits to a "zeno run" that must converge before time 5. ■

Given a real-time program $\mathcal{P}$, the guards and the invariant are state predicates, and the guarded commands assign the value of state predicates to propositions. It follows that the transition relation $S_{\mathcal{P}}$ can be characterized syntactically as a predicate transformer that maps state predicates to state predicates.

Definition 3.5 (precondition) For a state predicate ϕ and a guarded command $g = \psi \rightarrow A$, the *weakest precondition* of ϕ with respect to g is the state predicate

$$pre_g(\phi) \; = \; \psi \wedge \phi[A]$$

that characterizes all states from which a state satisfying ϕ can be obtained by executing the guarded command g. For a real-time program $\mathcal{P} = (G, \phi^{\Box})$, the *weakest precondition* of ϕ with respect to $\mathcal{P}$ is the state predicate

$$pre_{\mathcal{P}}(\phi) \; = \; \phi^{\Box} \wedge (\phi \vee \bigvee_{g \in G} pre_g(\phi^{\Box} \wedge \phi))$$

that characterizes all $\phi^{\Box}$-states from which a $\phi^{\Box}$-state satisfying ϕ can be obtained by either doing nothing or executing some guarded command of $\mathcal{P}$. ■

Proposition 3.2 *Let $\mathcal{P}$ be a real-time program and let ϕ be a state predicate. For all states $\sigma \in \Sigma$, $\sigma \models pre_{\mathcal{P}}(\phi)$ iff there exists a state $\sigma' \in \Sigma$ such that $(\sigma, \sigma') \in S_{\mathcal{P}}$ and $\sigma' \models \phi$.*

We will use the state predicate transformer $pre_{\mathcal{P}}$ to execute a nonzeno real-time program $\mathcal{P}$ symbolically.

3.4 Timed safety automata

The real-time program $\mathcal{P}_1$ of Example 3.3 is represented graphically in Figure 2. The graph of Figure 2 can be viewed as the transition diagram of a *timed automaton* [AD90]—a finite automaton with a finite set of real-valued clocks. We now define a class of timed automata—timed safety automata—for specifying divergence-safe real-time systems.[3]

Definition 3.6 (timed safety automaton: syntax) A *timed safety automaton* $\mathcal{A} = (L, E, f, g)$ is a labeled directed multigraph consisting of

1. L—a finite set of vertices called *locations*.
2. E—a multiset of edges $e \in L^2$ called *transitions*.
3. f—a vertex labeling function that assigns to each location $\ell \in L$ a past-closed state predicate $f(\ell)$, the *location invariant*.
4. g—an edge labeling function that assigns to each transition e in E a guarded command $g(e)$.

■

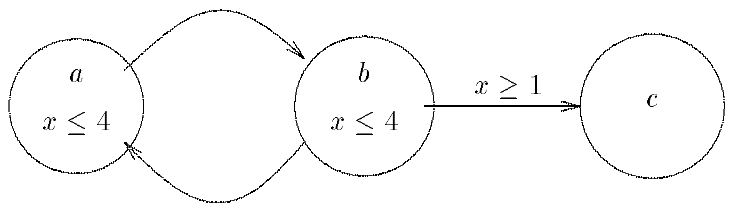

Figure 2: The timed safety automaton $\mathcal{A}_1$

Example 3.4 The timed safety automaton of Figure 2 has 3 locations, a, b, and c. The location invariant *true*, the guard *true*, and the assignment $x := x$ are suppressed. ■

[3]The reader familiar with [AD90] will notice that we use a nonstandard variety of timed automata that (1) permits clock constraints on both locations and transitions and (2) is interpreted over weakly monotonic time [AH92b]. Furthermore, to ensure that a timed automaton defines a divergence-safe real-time system, we assume that all locations of the automaton are Büchi accepting.

Instead of defining the transition relation $S_{\mathcal{A}}$ of a timed safety automaton $\mathcal{A}$ directly, we translate the automaton into a real-time program. For this purpose, we use a new proposition p_ℓ for each location $\ell \in L$ of the automaton $\mathcal{A}$. We write $\alpha = \ell$ for the state predicate $p_\ell \wedge \bigwedge_{m \in L - \{\ell\}} \neg p_m$, and $\alpha := \ell$ for the set $\{p_\ell := \mathit{true}\} \cup \{p_m := \mathit{false} \mid m \in L - \{\ell\}\}$ of assignments.

Definition 3.7 (timed safety automaton: semantics) Given a timed safety automaton $\mathcal{A} = (L, E, f, g)$, we define a real-time program $\mathcal{P}_{\mathcal{A}} = (G, \phi^{\Box})$: for each transition $e = (\ell_1, \ell_2)$ in E, if e is labeled with the guarded command $g(e) = \psi \rightarrow A$, then G contains the guarded command

$$(\alpha = \ell_1 \wedge \psi) \rightarrow A \cup \{\alpha := \ell_2\},$$

and the program invariant $\phi^{\Box}$ is the (past-closed) state predicate

$$\bigvee_{\ell \in L} (\alpha = \ell \wedge f(\ell)).$$

The timed safety automaton $\mathcal{A}$ defines the transition relation $S_{\mathcal{A}} = S_{\mathcal{P}_{\mathcal{A}}}$. We write $\Pi_{\mathcal{A}} = \Pi_{\mathcal{P}_{\mathcal{A}}}$ for the set of runs of $\mathcal{A}$. ∎

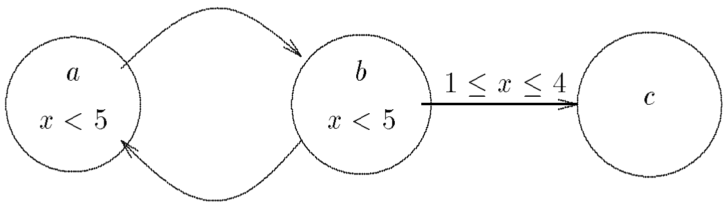

Figure 3: The timed safety automaton $\mathcal{A}_2$

Example 3.5 The zeno timed safety automaton $\mathcal{A}_2$ of Figure 3 corresponds to the zeno real-time program $\mathcal{P}_2 = \mathcal{P}_{\mathcal{A}_2}$ of Example 3.3. ∎

Conversely, it is not difficult to translate any real-time program $\mathcal{P}$ into a timed safety automaton $\mathcal{A}_{\mathcal{P}}$ with a single location and a transition for each guarded command of $\mathcal{P}$, such that $\mathcal{P}$ and $\mathcal{A}_{\mathcal{P}}$ define the same transition relation.

4 Real-time Logics

We present and compare two branching-time logics for specifying properties of real-time systems—the timed computation tree logic TCTL and the timed μ-calculus Tμ. Both languages use clocks and arithmetic constraints on clock values to specify timing requirements.

4.1 Timed computation tree logic

Many important properties of systems find a natural expression in temporal logic [Eme90]. We first review the real-time temporal logic TCTL [ACD90], which extends the branching-time logic CTL [EC82] with clock variables. The formulas of TCTL, which are interpreted over the states of a given premodel Π, are built from state predicates by boolean connectives, the two temporal until operators $\exists\mathcal{U}$ (*possibly*) and $\forall\mathcal{U}$ (*inevitably*), and a reset quantifier for clocks. Intuitively, the formula $p\exists\mathcal{U}q$ ($p\forall\mathcal{U}q$) holds in a state σ of Π iff the proposition q becomes true on some (every) trajectory in Π that starts from σ, and the proposition p is true until q becomes true.

The formulas of TCTL contain three kinds of variables. Propositional variables from P and clock variables from C occur freely and refer to the states of the given premodel Π. In addition, we may bind certain clock variables by reset quantifiers to express the timing requirements of a specification. A *specification clock* $z \in C$ is a clock that does not control the behavior of any system under consideration; that is, we consider only trajectories $\bar{\sigma}$ such that for all positions π of $\bar{\sigma}$,

$$\bar{\sigma}(\pi)(z) \;=\; \bar{\sigma}(0,0)(z) + \tau_{\bar{\sigma}}(\pi).$$

We write $C_\varphi \subseteq C$ for the set of specification clocks. The *reset quantifier* $z.$, which binds and resets the specification clock z, is inspired by the freeze quantifier of Timed Temporal Logic [AH89]: for example, the formula

$$z.(true\forall\mathcal{U}(q \wedge z \leq 5))$$

asserts that the proposition q becomes necessarily true within 5 time units.

Definition 4.1 (syntax of TCTL) The *formulas* φ of the timed computation tree logic TCTL are defined inductively by the grammar

$$\varphi \;::=\; p \mid x + c \leq y + d \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\exists\mathcal{U}\varphi_2 \mid \varphi_1\forall\mathcal{U}\varphi_2 \mid z.\varphi$$

for propositions $p \in P$, clocks $x, y \in C$, specification clocks $z \in C_\varphi$, and nonnegative integer constants $c, d \in \mathsf{N}$. ■

Additional arithmetic, boolean, and temporal operators, such as $=$, $\wedge$, and $\exists\Diamond$, are defined as usual. In particular, the temporal operators $\exists\Diamond\varphi$ and $\forall\Box\varphi$ stand for $true\exists\mathcal{U}\varphi$ and $\neg\exists\Diamond\neg\varphi$, respectively, and the temporal operators $\forall\Diamond\varphi$ and $\exists\Box\varphi$ stand for $true\forall\mathcal{U}\varphi$ and $\neg\forall\Diamond\neg\varphi$, respectively. Also note that all state predicates are definable in TCTL; for example, the state predicate $x \leq 5$ abbreviates the TCTL-formula $z.(x \leq z + 5)$.

The formulas of TCTL are interpreted over the states of a premodel. The propositions and the free clock variables of a TCTL-formula φ are evaluated in states. To evaluate bound clock variables, we use clock environments. A *clock environment* $\mathcal{E}$ is a partial function from the set C_φ of specification clocks to the nonnegative reals R^+. The *empty clock environment* $\emptyset$ is undefined on all specification clocks. The clock environment $\mathcal{E} + \delta$, for a delay $\delta \in \mathsf{R}^+$, maps the clock $z \in C_\varphi$ to the value $\mathcal{E}(z) + \delta$ if the environment $\mathcal{E}$ is defined on z; otherwise $\mathcal{E} + \delta$ is undefined on z. We write $\mathcal{E}[v := a]$ for the environment that agrees with the environment $\mathcal{E}$ on all variables except v, which is mapped to the value a.

Definition 4.2 (semantics of TCTL) Let Π be a premodel and let $\sigma \in \Sigma_\Pi$ be a state that is reachable in Π. The state σ *satisfies* the TCTL-formula φ in Π, denoted by $\sigma \models_\Pi \varphi$, if $\sigma \models_{\Pi,\emptyset} \varphi$ for the empty clock environment $\emptyset$. The satisfaction relation $\models_{\Pi,\mathcal{E}}$ is defined inductively for all clock environments $\mathcal{E}$:

$\sigma \models_{\Pi,\mathcal{E}} p$ iff $\sigma(p) = true$;

$\sigma \models_{\Pi,\mathcal{E}} x + c \leq y + d$ iff $f(x) + c \leq f(y) + d$,

where $f(v) = \mathcal{E}(v)$ if $\mathcal{E}$ is defined on v, and $f(v) = \sigma(v)$ otherwise;

$\sigma \models_{\Pi,\mathcal{E}} \neg\varphi$ iff $\sigma \not\models_{\Pi,\mathcal{E}} \varphi$;

$\sigma \models_{\Pi,\mathcal{E}} \varphi_1 \vee \varphi_2$ iff $\sigma \models_{\Pi,\mathcal{E}} \varphi_1$ or $\sigma \models_{\Pi,\mathcal{E}} \varphi_2$;

$\sigma \models_{\Pi,\mathcal{E}} \varphi_1 \exists\mathcal{U} \varphi_2$ iff for some trajectory $\bar{\sigma} \in \Pi$ with $\bar{\sigma}(0,0) = \sigma$,

there exists a position (i, δ) of $\bar{\sigma}$ such that $\bar{\sigma}(i, \delta) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(i,\delta)} \varphi_2$ and
for all positions (j, ϵ) of $\bar{\sigma}$, if $(j, \epsilon) \prec (i, \delta)$ then $\bar{\sigma}(j, \epsilon) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(j,\epsilon)} \varphi_1 \vee \varphi_2$;

$\sigma \models_{\Pi,\mathcal{E}} \varphi_1 \forall\mathcal{U} \varphi_2$ iff for all trajectories $\bar{\sigma} \in \Pi$ with $\bar{\sigma}(0,0) = \sigma$,

there exists a position (i, δ) of $\bar{\sigma}$ such that $\bar{\sigma}(i, \delta) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(i,\delta)} \varphi_2$ and
for all positions (j, ϵ) of $\bar{\sigma}$, if $(j, \epsilon) \prec (i, \delta)$ then $\bar{\sigma}(j, \epsilon) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(j,\epsilon)} \varphi_1 \vee \varphi_2$;

$\sigma \models_{\Pi,\mathcal{E}} z.\,\varphi$ iff $\sigma \models_{\Pi,\mathcal{E}[z:=0]} \varphi$.

■

We point out that the density of the time domain causes a subtle complication in the definition of both until operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$. Suppose that the formula "ϕ_1 until ϕ_2" holds on a trajectory $\bar{\sigma}$ iff ϕ_2 becomes true in $\bar{\sigma}$ and ϕ_1 is true in $\bar{\sigma}$ until ϕ_2 becomes true. In particular, the state predicate ϕ_2 may become true either *at* a position π of $\bar{\sigma}$ or in a left-open interval immediately *following* a position π of $\bar{\sigma}$ (compare the formulas "$x < 5$ until $x \geq 5$" and "$x \leq 5$ until $x > 5$"). To account for both possibilities, we require that there is a position π of $\bar{\sigma}$ at which ϕ_2 is true and that the disjunction $\phi_1 \vee \phi_2$—rather than ϕ_1 only—is true at all positions of $\bar{\sigma}$ before π. It follows that the TCTL-formula $z.((z \leq 5)\forall\mathcal{U}(z > 5))$ holds in each state of every real-time system.

A few comments regarding our particular version of TCTL are in order. First, unlike [ACD90], we have opted for weakly monotonic time. Second, the time-bounded temporal operators used in [ACD90] are definable in TCTL. For instance, the time-bounded response requirement

$$\forall\Box(p \rightarrow \forall\Diamond_{\leq 5}\, q),$$

which asserts that every request p must be followed by a response q within 5 time units, is expressible in TCTL by the formula

$$\forall\Box z.\,(p \rightarrow \forall\Diamond(q \wedge z \leq 5)).$$

Thus we will use time-bounded temporal operators such as $\forall\Diamond_{\leq 5}$ as abbreviations. Third, in our version of TCTL freeze quantifiers on static time variables are replaced by reset quantifiers on dynamic clock variables. While both approaches are equivalent [Alu91], the clock-reset style fits in better with our definition of real-time programs.

4.2 Finite variability

Given a premodel Π, every formula φ of TCTL defines a set $s \subseteq \Sigma_\Pi$ of reachable states, namely, those states that satisfy φ. We will refer to s sometimes as the characteristic set of φ (in Π) and sometimes as the requirement (of Π) specified by φ. The following definitions apply to all logics, such as TCTL, whose formulas are interpreted over the states of a premodel.

Definition 4.3 (characteristic set) The *characteristic set* $[\![\varphi]\!]_\Pi$ of a formula φ in a premodel Π is the set of all reachable states that satisfy φ in Π:

$$[\![\varphi]\!]_\Pi \;=\; \{\sigma \in \Sigma_\Pi \mid \sigma \models_\Pi \varphi\}.$$

■

When interpreting a formula over a premodel Π, it will often be convenient to consider, in place of a subformula, a set $s \subseteq \Sigma$ of states. For example, we write $[\![\exists\Diamond s]\!]_\Pi$ for the set of states from which a trajectory in Π leads to a state in s. Formally, for all states $\sigma \in \Sigma_\Pi$ and all environments $\mathcal{E}$, let $\sigma \models_{\Pi,\mathcal{E}} s$ iff $\sigma \in s$.

Definition 4.4 (finite variability) Given a trajectory $\bar{\sigma}$ and a nonnegative integer i, recall that $s_i(\bar{\sigma})$ stands for the set of states in the i-th segment of $\bar{\sigma}$ (Section 2.1). The trajectory $\bar{\sigma}$ *respects* a set $s \subseteq \Sigma$ of states if for all $i \in \mathsf{N}$, either $s_i(\bar{\sigma}) \subseteq s$ or $s_i(\bar{\sigma}) \cap s = \emptyset$. The set s is *finitely variable* along $\bar{\sigma}$ if there is a trajectory $\bar{\sigma}' \in \Gamma(\bar{\sigma})$ in the stutter closure of $\bar{\sigma}$ such that $\bar{\sigma}'$ respects s. The set s is *finitely variable* over a set Π of trajectories if s is finitely variable over all trajectories in Π. A logic is *finitely variable* over a class $\mathcal{C}$ of premodels if for all formulas φ and all premodels $\Pi \in \mathcal{C}$, the characteristic set $[\![\varphi]\!]_\Pi$ is finitely variable over Π. ■

Let $\bar{\sigma}$ be a trajectory. We defined trajectories so that the characteristic sets of all atomic state predicates are finitely variable along $\bar{\sigma}$. Moreover, the state sets that are finitely variable along $\bar{\sigma}$ are closed under complement, union, and finite intersection. It follows that all state predicates define finitely variable sets.

Proposition 4.1 *The logic of state predicates is finitely variable over all premodels.*

TCTL, however, is not finitely variable over all premodels. The following example presents a TCTL-formula φ and a divergence-safe real-time system Π_3 such that the characteristic set $[\![\varphi]\!]_{\Pi_3}$ is not finitely variable over Π_3.

Example 4.1 Let Π_3 be the real-time system that first changes the value of a proposition p from *false* to *true* at time $2 - \frac{1}{2m}$, for some positive integer $m > 0$, and then changes the value of p back to *false* at time $2 - \frac{1}{2m+1}$. More precisely, Π_3 is the set of divergent S-trajectories, where the transition relation S is the reflexive closure of the binary relation

$$\left\{(\sigma,\sigma') \mid \exists m > 0.\left(\begin{array}{l} (\sigma(p) = \mathit{false} \wedge \sigma(x) = 2 - \frac{1}{2m} \wedge \sigma'(p) = \mathit{true} \wedge \sigma'(x) = \sigma(x)) \vee \\ (\sigma(p) = \mathit{true} \wedge \sigma(x) = 2 - \frac{1}{2m+1} \wedge \sigma'(p) = \mathit{false} \wedge \sigma'(x) = \sigma(x)) \end{array}\right)\right\}.$$

Then Π_3 is divergence-safe and the TCTL-formula $\exists\Diamond_{=1} p$ is not finitely variable over Π_3:

$$[\![\exists\Diamond_{=1} p]\!]_{\Pi_3} \;=\; \{\sigma \mid \exists m > 0.\,(\sigma(p) = \mathit{false} \wedge \frac{1}{2m+1} \leq 1 - \sigma(x) \leq \frac{1}{2m})\}.$$

■

The branching structure of the real-time system Π_3 cannot be specified by a real-time program. Indeed, if we restrict our attention to real-time systems that are definable by guarded-command real-time programs (or timed safety automata), then TCTL is finitely variable. This is because over real-time programs, all state sets that can be defined by TCTL-formulas can already be defined by state predicates. These state sets are called regions.

Definition 4.5 (region) Given a real-time program $\mathcal{P}$ and a formula φ, let $C_\mathcal{P} \subseteq C$ be the set of clocks that occur in $\mathcal{P}$ (within a guard or within the program invariant) or freely in φ, and let k be the largest integer constant that occurs in $\mathcal{P}$ or in φ. A set of states $s \subseteq \Sigma$ is a $(\mathcal{P},\varphi)$-*region* if there is a state predicate ϕ such that $s = [\![\phi]\!]$, all clock variables in ϕ are from $C_\mathcal{P}$, and the largest integer constant in ϕ is at most k. A $(\mathcal{P},\varphi)$-region is *minimal* if it does not properly contain a nonempty $(\mathcal{P},\varphi)$-region. ■

Suppose, for example, that $C_{\mathcal{P}} = \{x_1, x_2, x_3\}$ and $k = 5$. Then the state predicate

$$\phi \;=\; (1 < x_1 < 2 \,\wedge\, x_2 = 3 \,\wedge\, 4 < x_3 < 5)$$

defines a region that is the disjoint union of three minimal regions, namely, $[\![\phi \,\wedge\, x_1 + 3 < x_3]\!]$, $[\![\phi \,\wedge\, x_1 + 3 = x_3]\!]$, and $[\![\phi \,\wedge\, x_1 + 3 > x_3]\!]$.

Some observations about regions are immediate. Let $\mathcal{P}$ be a real-time program and let φ be a TCTL-formula. First, there are only finitely many $(\mathcal{P}, \varphi)$-regions: if $\mathcal{P}$ and φ contain n clocks and no constant larger than k, then there are $O(n^k \cdot n!)$ minimal $(\mathcal{P}, \varphi)$-regions [Alu91]; and every $(\mathcal{P}, \varphi)$-region is a finite disjoint union of minimal $(\mathcal{P}, \varphi)$-regions. Second, the $(\mathcal{P}, \varphi)$-regions are closed under all boolean operations. Third, since the real-time system $\Pi_{\mathcal{P}}$ is closed under past, Theorem 3.2 implies the following lemma, which asserts that the $(\mathcal{P}, \varphi)$-regions are closed under time delays.

Lemma 4.1 *Let $\mathcal{P}$ be a real-time program, let φ be a* TCTL*-formula, and let ϕ_1 and ϕ_2 be two state predicates. If $[\![\phi_1]\!]$ and $[\![\phi_2]\!]$ are $(\mathcal{P}, \varphi)$-regions, then $[\![\phi_1 \rightsquigarrow \phi_2]\!]$ is a $(\mathcal{P}, \varphi)$-region.*

We now restate in our framework the main theorem for timed automata from Alur's thesis. By Proposition 4.1, we obtain the finite variability of TCTL over real-time programs as a corollary.

Theorem 4.1 [Alu91] *For every real-time program $\mathcal{P}$ and every* TCTL*-formula φ, the characteristic sets $[\![\varphi]\!]_{\Pi_{S_{\mathcal{P}}}}$ and $[\![\varphi]\!]_{\Pi_{\mathcal{P}}}$ are $(\mathcal{P}, \varphi)$-regions.*

Corollary 4.1 TCTL *is finitely variable over the class of real-time programs.*

4.3 The timed μ-calculus

We now introduce Tμ, a dense-time μ-calculus with clocks. The formulas of Tμ are built from state predicates by boolean connectives, a temporal next operator, the reset quantifier for clocks, and a least-fixpoint quantifier. While discrete-time μ-calculi rely on a unary next-time operator [Koz83, Eme92], there is no notion of "next time" when time is modeled by the real numbers. Instead, we use the binary *next operator* $\triangleright$, which is best viewed as a "single-step until" operator of temporal logic:[4] roughly speaking, the formula $p \triangleright q$ asserts that p is true now and stays true until some transition is taken, and this transition establishes q. In particular, the "next" transition may be arbitrarily close in time or arbitrarily far away. For example, the formula

$$z.\,(\mathit{true} \triangleright (q \,\wedge\, z \leq 5))$$

asserts that q can be established by a single transition taken within 5 time units.

The formulas of Tμ contain four kinds of variables. In addition to free propositional variables, free clock variables, and specification clocks that are bound by reset quantifiers, Tμ-formulas may contain formula variables that are bound by least-fixpoint quantifiers. We use V as the set of formula variables and write $\mu X.$ for a *least-fixpoint quantifier* that binds the formula variable X.

Definition 4.6 (syntax of Tμ) The *formulas* φ of the timed μ-calculus Tμ are defined inductively by the grammar

$$\varphi \;::=\; X \mid p \mid x + c \leq y + d \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \triangleright \varphi_2 \mid z.\,\varphi \mid \mu X.\,\varphi$$

[4] A modal operator similar to our next operator has been proposed for a dense-time extension of Hennessy-Milner logic [HLW91].

for formula variables $X \in V$, propositions $p \in P$, clocks $x, y \in C$, specification clocks $z \in C_\varphi$, and nonnegative integer constants $c, d \in \mathsf{N}$. We require that every occurrence of a formula variable $X \in V$ in φ is bound (i.e, it appears within the scope of a least-fixpoint quantifier $\mu X.$) and positive (i.e., it appears within an even number of negations from the quantifier $\mu X.$ that binds X). ■

The standard condition that all formula variables must occur positively ensures that the arguments of all least-fixpoint quantifiers define monotonic functionals, which guarantees the existence of the corresponding fixpoints. Additional arithmetic and boolean operators are defined as usual and, as in the case of TCTL, all state predicates are definable in Tμ. We write $\varphi[X := a]$ for the Tμ-formula that results from φ by replacing each free occurrence of the formula variable X with the expression a. The abbreviation $\nu X.\,\varphi$ denotes a *greatest-fixpoint quantifier* and stands for the Tμ-formula $\neg\mu X.\,(\neg\varphi[X := \neg X])$.

The formulas of Tμ are interpreted over the states Σ of a given premodel. For this purpose, the environment needs to be extended to provide values for both specification clocks and formula variables. A *Tμ-environment* $\mathcal{E}$ consists of a clock environment $\mathcal{E}_1$ together with a total function $\mathcal{E}_2$ from the set V of formula variables to the power set 2^Σ of states; that is, for all specification clocks $z \in C_\varphi$, either $\mathcal{E}(z) \in \mathsf{R}^+$ or $\mathcal{E}(z)$ is undefined, and for all formula variables $X \in V$, $\mathcal{E}(X) \subseteq \Sigma$. The T$\mu$-environment $\mathcal{E}$ is *empty* if its clock component is empty (i.e., $\mathcal{E}_1 = \emptyset$). The T$\mu$-environment $\mathcal{E} + \delta$, for a delay $\delta \in \mathsf{R}^+$, consists of the clock component $\mathcal{E}_1 + \delta$ and the formula-variable component $\mathcal{E}_2$.

Definition 4.7 (semantics of Tμ) Let Π be a premodel and let $\sigma \in \Sigma_\Pi$ be a state that is reachable in Π. The state σ *satisfies* the Tμ-formula φ in Π, denoted by $\sigma \models_\Pi \varphi$, if $\sigma \models_{\Pi,\mathcal{E}} \varphi$ for all empty Tμ-environments $\mathcal{E}$. The satisfaction relation $\models_{\Pi,\mathcal{E}}$ is defined inductively for all Tμ-environments $\mathcal{E}$; only the following clauses differ from the definition of the semantics for TCTL:

$\sigma \models_{\Pi,\mathcal{E}} X$ iff $\sigma \in \mathcal{E}(X)$;

$\sigma \models_{\Pi,\mathcal{E}} \varphi_1 \triangleright \varphi_2$ iff for some state $\sigma' \in \Sigma_\Pi$ and some delay $\delta \in \mathsf{R}^+$,

$\sigma \triangleright^{\delta}_{S_\Pi} \sigma'$ and $\sigma' \models_{\Pi,\mathcal{E}+\delta} \varphi_2$, and
for all delays $\epsilon \in \mathsf{R}^+$, if $\epsilon \leq \delta$ then $\sigma + \epsilon \models_{\Pi,\mathcal{E}+\epsilon} \varphi_1 \vee \varphi_2$;

$\sigma \models_{\Pi,\mathcal{E}} \mu X.\,\varphi$ iff $\sigma \in \bigcap\{s \subseteq \Sigma_\Pi \mid \{\sigma' \in \Sigma_\Pi \mid \sigma' \models_{\Pi,\mathcal{E}[X:=s]} \varphi\} = s\}$.

For a formula φ of Tμ, we write $[\![\varphi]\!]_\Pi \subseteq \Sigma_\Pi$ for the set of reachable states that satisfy φ. ■

Let Π be a premodel and let $\mathcal{E}$ be a Tμ-environment. Now consider the subformula $\mu X.\,\varphi$ of a Tμ-formula; that is, $\mu X.\,\varphi$ may contain free formula variables. The semantic clause for the least-fixpoint quantifier ensures that the formula $\mu X.\,\varphi$ defines the least fixpoint of the functional $F_\mathcal{E}(X)$ from 2^{Σ_Π} to 2^{Σ_Π}: for $s \subseteq \Sigma_\Pi$ and $\sigma \in \Sigma_\Pi$, $\sigma \in F_\mathcal{E}(s)$ iff $\sigma \models_{\Pi,\mathcal{E}[X:=s]} \varphi$. The least fixpoint of the functional $F_\mathcal{E}$ exists by the Tarski-Knaster theorem, because $F_\mathcal{E}$ is a monotonic functional on the complete lattice $(2^{\Sigma_\Pi}, \subseteq)$ (i.e., $s_1 \subseteq s_2$ implies $F_\mathcal{E}(s_1) \subseteq F_\mathcal{E}(s_2)$). To see this, recall that all free occurrences of X in φ are positive and observe that disjunction, conjunction, and, by the following lemma, the next operator are monotonic in both arguments.

Lemma 4.2 *Let Π be a premodel. For all sets $s, s_1, s_2 \subseteq \Sigma_\Pi$, if $s_1 \subseteq s_2$ then $[\![s_1 \triangleright s]\!]_\Pi \subseteq [\![s_2 \triangleright s]\!]_\Pi$ and $[\![s \triangleright s_1]\!]_\Pi \subseteq [\![s \triangleright s_2]\!]_\Pi$.*

The semantic clause for the next operator $\triangleright$ uses existential quantification over successor states. The dual operator, a unary next operator with universal force, is definable in Tμ: for all premodels Π, all Tμ-environments $\mathcal{E}$, and all reachable states $\sigma \in \Sigma_\Pi$,

$$\sigma \models_{\Pi,\mathcal{E}} \neg(\mathit{true} \triangleright (\neg\varphi))$$

iff for all states $\sigma' \in \Sigma_\Pi$ and all delays $\delta \in \mathsf{R}^+$, if $\sigma \triangleright^{\delta}_{S_\Pi} \sigma'$ then $\sigma' \models_{\Pi,\mathcal{E}+\delta} \varphi$ (the stutter closure of Π implies, then, that $\sigma + \epsilon \models_{\Pi,\mathcal{E}+\epsilon} \varphi$ for all delays $0 \leq \epsilon \leq \delta$).

4.4 Computing fixpoints

Let Π be a premodel and let $\mu X.\, \varphi$ be a formula of Tμ. Since the characteristic set $[\![\mu X.\, \varphi]\!]_\Pi$ is the least fixpoint of the monotonic functional $F(X) = [\![\varphi]\!]_\Pi$, it follows that

$$[\![\mu X.\, \varphi]\!]_\Pi \;=\; \bigcup_{i \in \mathcal{O}} F^i(\emptyset)$$

for a sufficiently large ordinal $\mathcal{O}$; that is, the characteristic set $[\![\mu X.\, \varphi]\!]_\Pi$ is the limit of the successive approximation sequence

$$\emptyset, F(\emptyset), F^2(\emptyset), F^3(\emptyset), \ldots$$

We now show that, over the class of real-time programs, all elements of this sequence are regions. Indeed, we construct the state predicates that define the regions.

The following lemma is fundamental for computing fixpoints over both zeno and nonzeno real-time programs by successive approximation. To accommodate nested fixpoint quantifiers, we need to look at Tμ-formulas with several free formula variables.

Lemma 4.3 *Let $\mathcal{P}$ be a real-time program, let φ be a fixpoint-quantifier-free subformula of a Tμ-formula with the formula variables $X_1, \ldots, X_n$, and let $A = \{X_1 := s_1, \ldots, X_n := s_n\}$ be a set of assignments of state sets to formula variables. If the state sets $s_1, \ldots, s_n$ are $(\mathcal{P}, \varphi)$-regions, then so is the characteristic set $[\![\varphi[A]]\!]_{\Pi_{S_\mathcal{P}}}$. Moreover, if the real-time program $\mathcal{P}$ is nonzeno, then $[\![\varphi[A]]\!]_{\Pi_\mathcal{P}} = [\![\varphi[A]]\!]_{\Pi_{S_\mathcal{P}}}$.*

Proof. Let $\phi^\Box$ be the invariant of the real-time program $\mathcal{P}$. We use induction on the structure of the subformula φ. Throughout the proof, let $\Pi = \Pi_{S_\mathcal{P}}$ and, if $\mathcal{P}$ is nonzeno, $\Pi = \Pi_\mathcal{P}$.

Suppose that φ is the formula variable X_i and the region s_i is defined by the state predicate ϕ_i. From Proposition 3.1 it follows that $[\![\varphi[A]]\!]_\Pi$ is defined by the state predicate $\phi^\Box \wedge \phi_i$.

If φ is a proposition or a clock constraint, then $[\![\varphi[A]]\!]_\Pi$ is defined by the state predicate $\phi^\Box \wedge \varphi$.

The boolean cases follow from the fact that the $(\mathcal{P}, \varphi)$-regions are closed under all boolean operations.

If φ is of the form $z.\, \psi$ and $[\![\psi[A]]\!]_\Pi$ is defined by the state predicate ϕ, then $[\![\varphi[A]]\!]_\Pi$ is defined by the state predicate $\phi[z := 0]$.

Finally, suppose that φ is of the form $\psi_1 \triangleright \psi_2$ and the characteristic sets $[\![\psi_i[A]]\!]_\Pi$, for $i = 1, 2$, are defined by the state predicates ϕ_i. From Proposition 3.2 it follows that $[\![\varphi[A]]\!]_\Pi$ is defined by the evolving state predicate

$$(\phi_1 \vee \phi_2) \rightsquigarrow \mathit{pre}_\mathcal{P}(\phi_2).$$

The lemma follows from Theorem 3.2. ■

Let $\mathcal{P}$ be a nonzeno real-time program. Since there are only finitely many $(\mathcal{P}, \varphi)$-regions,

$$[\![\mu X.\, \varphi]\!]_{\Pi_\mathcal{P}} \;=\; \bigcup_{0 \leq i \leq m} F^i(\emptyset)$$

for a sufficiently large integer m; that is, the limit of the successive approximation sequence is reached within a finite number of steps. This observation is the basis for the symbolic model-checking algorithm presented in Section 6.1. More precisely, the number m of iterations of the functional F that are necessary to compute the least fixpoint of F is bounded by the number of minimal $(\mathcal{P}, \varphi)$-regions, which is exponential in the representations of $\mathcal{P}$ and φ. In general, the number of iterations that are required by this method to compute the characteristic set $[\![\varphi]\!]_{\Pi_\mathcal{P}}$ of an arbitrary Tμ-formula φ depends exponentially not only on the number of clocks and the size of the largest constant in $\mathcal{P}$ and φ, but also on the nesting depth of least-fixpoint quantifiers in φ.

In Section 6.3 we will show that for every real-time program, there is an equivalent nonzeno program. The theorem below follows.

Theorem 4.2 *For every real-time program $\mathcal{P}$ and every* Tμ*-formula φ, the characteristic sets $[\![\varphi]\!]_{\Pi_{S_\mathcal{P}}}$ and $[\![\varphi]\!]_{\Pi_\mathcal{P}}$ are $(\mathcal{P}, \varphi)$-regions.*

Corollary 4.2 Tμ *is finitely variable over the class of real-time programs.*

By contrast, Tμ is not finitely variable over all premodels (consider the divergence-safe real-time system Π_3 of Example 4.1 and the Tμ-formula $z.(\mathit{true} \triangleright (p \wedge z = 1))$).

5 The Expressive Power of Fixpoints

We now compare the expressive power of the timed μ-calculus Tμ with the expressive power of the timed computation tree logic TCTL. The expressive power of a logic is measured by determining which requirements of a given class of premodels are definable in the logic.

Definition 5.1 (expressiveness) Let $\mathcal{C}$ be a class of premodels. The formula φ_A is *equivalent* to the formula φ_B over the class $\mathcal{C}$ if $[\![\varphi_A]\!]_\Pi = [\![\varphi_B]\!]_\Pi$ for all premodels $\Pi \in \mathcal{C}$. The logic $\mathcal{A}$ is *as expressive as* the logic $\mathcal{B}$ over the class $\mathcal{C}$ if for every $\mathcal{B}$-formula φ_B there is an $\mathcal{A}$-formula φ_A that is equivalent to φ_B over $\mathcal{C}$. ■

Depending on the class of premodels under consideration, we obtain different results about the relative expressiveness of two logics. In particular, we will show that while some TCTL-requirements of real-time systems cannot be specified in Tμ, all TCTL-requirements of guarded-command real-time programs can be specified in Tμ. Recall that TCTL is based on the temporal until operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$, and Tμ is based on the temporal next operator $\triangleright$. We proceed in three steps. First, we show that the possibility operator $\exists\mathcal{U}$ is definable as a fixpoint over all premodels. Second, we show that the inevitability operator $\forall\mathcal{U}$ is not definable as a fixpoint over all real-time systems. Third, we show that the inevitability operator $\forall\mathcal{U}$ is definable as a fixpoint over safe premodels and, for finitely variable arguments, also over divergence-safe real-time systems.

5.1 Possibility over premodels

Let us begin with a couple of positive results. We show that, over all premodels, the possibility (reachability) operator $\exists\mathcal{U}$ is definable in Tμ, and the next operator $\triangleright$ is definable in TCTL.

Proposition 5.1 *For all premodels Π and all sets $s_1, s_2 \subseteq \Sigma_\Pi$ of reachable states,*

$$[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi \;=\; [\![\mu X.(s_2 \vee (s_1 \triangleright X))]\!]_\Pi.$$

Proof. We first show that the set $[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi$ is a solution of the equation $X = [\![s_2 \vee (s_1 \triangleright X)]\!]_\Pi$; that is,

$$[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi \;=\; [\![s_2 \vee (s_1 \triangleright (s_1 \exists\mathcal{U} s_2))]\!]_\Pi.$$

Let $\sigma \in \Sigma_\Pi$ be a reachable state. Since $\sigma \triangleright^0_{S_\Pi} \sigma$, if $\sigma \models_\Pi s_1 \exists\mathcal{U} s_2$ and $\sigma \notin s_2$, then $\sigma \models_\Pi s_1 \triangleright (s_1 \exists\mathcal{U} s_2)$. Conversely, if $\sigma \in s_2$ then $\sigma \models_\Pi s_1 \exists\mathcal{U} s_2$; and since Π is future-closed and fusion-closed, if $\sigma \models_\Pi s_1 \triangleright (s_1 \exists\mathcal{U} s_2)$ then $\sigma \models_\Pi s_1 \exists\mathcal{U} s_2$.

It remains to be shown that the set $[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi$ is the least solution of the equation $X = [\![s_2 \vee (s_1 \triangleright X)]\!]_\Pi$. Let $s \subseteq \Sigma_\Pi$ be such that $s = [\![s_2 \vee (s_1 \triangleright s)]\!]_\Pi$; we show that $[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi \subseteq s$. Let $\sigma \in \Sigma_\Pi$ be such that $\sigma \models_\Pi s_1 \exists\mathcal{U} s_2$. Then there is a trajectory $\bar\sigma \in \Pi$ and a position (i, δ) of $\bar\sigma$ such that $\bar\sigma(0,0) = \sigma$, $\bar\sigma(i,\delta) \in s_2$, and for all positions (j,ϵ) of $\bar\sigma$, if $(j,\epsilon) \prec (i,\delta)$ then $\bar\sigma(j,\epsilon) \in (s_1 \cup s_2)$. By induction on $m \in \mathsf{N}$, it follows that $\bar\sigma(i-m, 0) \in s$ for all $0 \leq m \leq i$. In particular, $\sigma \in s$. ■

In other words, the characteristic set of the formula $s_1 \exists\mathcal{U} s_2$ is the least fixpoint of the functional

$$F(X) \;=\; [\![s_2 \vee (s_1 \triangleright X)]\!]_\Pi.$$

From the proof of Proposition 5.1 it follows, moreover, that

$$[\![s_1 \exists\mathcal{U} s_2]\!]_\Pi \;=\; \bigcup_{i \in \mathsf{N}} F^i(\emptyset).$$

Intuitively, the set $F^i(\emptyset)$ contains all states from which a state in s_2 can be reached by at most i transitions along a path in $s_1 \cup s_2$. We point out that the functional F, while monotonic (Lemma 4.2), is not necessarily continuous.[5]

The next operator $\triangleright$ is definable in TCTL, provided that the set $P \cup C$ of propositions and clocks is, as we have assumed, finite. To define the formula $\varphi_1 \triangleright \varphi_2$ using the possibility operator $\exists\mathcal{U}$, we must ensure in the first argument of $\exists\mathcal{U}$ that no proposition changes its value and no clock is reset (i.e., no clock changes from a positive value to the value 0). This assurance can be given by a finite disjunction of formulas of the form

$$(\phi_i \wedge \varphi_1) \exists\mathcal{U} (\varphi_2 \;\vee\; ((\phi'_i \wedge \varphi_1) \exists\mathcal{U} \varphi_2)),$$

where the state predicates ϕ_i enumerate all possible proposition values (boolean) and clock values (zero or positive), and the state predicate ϕ'_i agrees with ϕ_i on all propositions while asserting that all clocks are positive. It follows that for every TCTL-formula φ with next operators, there is a TCTL-formula without next operators that is equivalent to φ over all premodels.

5.2 Inevitability over real-time systems

The following observation is instrumental for characterizing the expressive power of fixpoints: the timed μ-calculus cannot distinguish between premodels that induce the same transition relation.

Proposition 5.2 *For all premodels Π_1 and Π_2 and all Tμ-formulas φ, if $S_{\Pi_1} = S_{\Pi_2}$ then $[\![\varphi]\!]_{\Pi_1} = [\![\varphi]\!]_{\Pi_2}$.*

Proof. First observe that $S_{\Pi_1} = S_{\Pi_2}$ implies $\Sigma_{\Pi_1} = \Sigma_{\Pi_2}$. It is straightforward to show by induction on the structure of φ that, for all reachable states $\sigma \in \Sigma_{\Pi_1}$ and all Tμ-environments $\mathcal{E}$, $\sigma \models_{\Pi_1,\mathcal{E}} \varphi$ iff $\sigma \models_{\Pi_2,\mathcal{E}} \varphi$. ■

[5] Consider a real-time system Π that contains only trajectories along which some clock x is never reset, including a trajectory that starts with $x = 0$. Let $s_1 = [\![x = 0]\!]_\Pi$, $s_2 = \emptyset$, and $X_i = [\![x \geq \frac{1}{i}]\!]_\Pi$ for all $i \geq 1$. Then $F(\bigcup_{i \geq 1} X_i) = \Sigma_\Pi$ and $\bigcup_{i \geq 1} F(X_i) = \emptyset$.

Corollary 5.1 *Let φ be a formula of* Tμ. *For all premodels* Π, $[\![\varphi]\!]_\Pi = [\![\varphi]\!]_{\overline{\Pi}}$*; and for all real-time systems* Π, $[\![\varphi]\!]_\Pi = [\![\varphi]\!]_{\widetilde{\Pi}}$.

We apply Proposition 5.2 to two examples. First, consider the TCTL-formula

$$\varphi_{div} \;=\; \forall\Box z.\forall\Diamond(z=1).$$

The formula φ_{div} is satisfied in a state σ of a premodel Π iff time diverges along all trajectories that start with σ. Consequently, $[\![\varphi_{div}]\!]_\Pi = \Sigma_\Pi$ iff the premodel Π is a real-time system. It is not difficult to see that no Tμ-formula is equivalent to the TCTL-formula φ_{div} over all premodels (note that the Tμ-formula *true* is trivially equivalent to φ_{div} over the class of real-time systems). For suppose that there is some Tμ-formula φ asserting that a premodel Π is a real-time system (i.e., $[\![\varphi]\!]_\Pi = \Sigma_\Pi$ iff Π is a real-time system), then by Corollary 5.1, $\overline{\Pi}$ is also a real-time system, which is a contradiction (no real-time system is safe). It follows that the timed μ-calculus Tμ is, over all premodels, not as expressive as the branching-time logic TCTL. Indeed, we will now strengthen this observation to the class of real-time systems.

Second, recall the real-time system Π_2 of Example 2.1, which eventually changes the value of the proposition p from *false* to *true*. The safety closure $\overline{\Pi}_2$ contains a trajectory $\bar{\sigma}$ such that $\bar{\sigma}(\pi)(p) = \mathit{false}$ for all positions π of $\bar{\sigma}$. Thus, while all reachable states of Π_2 satisfy the TCTL-formula $\forall\Diamond p$, this it not the case for $\overline{\Pi}_2$. From Corollary 5.1 it follows that the property $\forall\Diamond p$ cannot be defined in Tμ over the class of real-time systems. Hence we have the first part of the following result.

Theorem 5.1 Tμ *is not as expressive as* TCTL *over the class of real-time systems, and* TCTL *is not as expressive as* Tμ *over the class of real-time programs.*

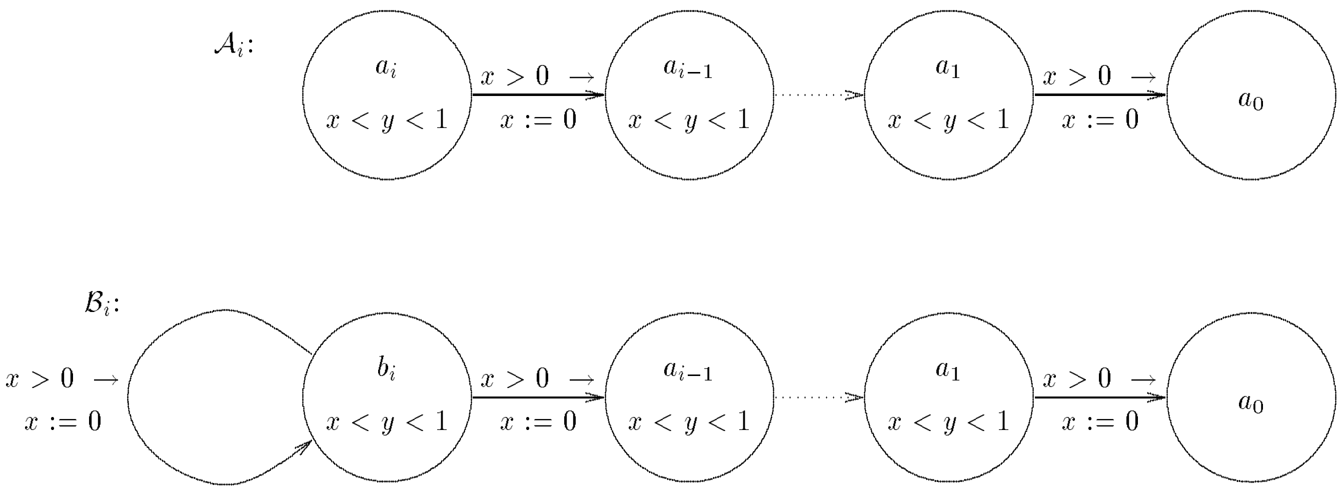

Figure 4: Timed safety automata $\mathcal{A}_i$ and $\mathcal{B}_i$

Proof. We need to show only the second part of Theorem 5.1. First recall that CTL is strictly less expressive than the propositional μ-calculus; in particular, the CTL*-formula

$$\exists\Box\Diamond p \;=\; \nu X.\exists\Diamond(p \wedge \exists\bigcirc X),$$

which asserts that there is a trajectory that contains infinitely many p-states, cannot be defined in CTL [EC80]. We modify a combinatorial proof of this result [EH86] to show that the Tμ-formula

$$\varphi_{inf} \;=\; \nu X.\, \exists\Diamond(x = 0 \;\wedge\; ((x = 0) \triangleright ((x > 0) \triangleright X)))$$

is not definable in TCTL over the class of real-time programs.

The proof proceeds by contradiction. Suppose that there is a TCTL-formula φ such that $[\![\varphi]\!]_{\Pi_{\mathcal{P}}} = [\![\varphi_{inf}]\!]_{\Pi_{\mathcal{P}}}$ for all real-time programs $\mathcal{P}$. Let $\hat{\imath}$ be nesting depth of temporal operators in φ, and let $\hat{P} \subseteq P$ be a set of propositions that do not occur in φ.

Now consider the two sequences $\mathcal{A}_i$ and $\mathcal{B}_i$, for $i \in \mathsf{N}$, of timed safety automata defined in Figure 4. The automata modify the control variable α, whose value $\sigma(\alpha) \in \{a_i, b_i \mid i \in \mathsf{N}\}$ is encoded using propositions from $\hat{P}$, and the two clocks x and y. We write $\hat{\Sigma} \subseteq \Sigma$ for the set of states σ with $0 \leq \sigma(x) < \sigma(y) < 1$. Given a state $\sigma \in \hat{\Sigma}$, a clock environment $\mathcal{E}$ is a σ-environment if $\mathcal{E}(z) < \sigma(y)$ for all specification clocks $z \in C_\varphi$. To avoid cumbersome notation, we write $\sigma[\ell]$ for the state $\sigma[\alpha := \ell]$ and suppress automaton designators whenever possible (note that every $\mathcal{A}_i$-run that starts from a state $\sigma[a_j]$, where $i \geq j$, is a $\mathcal{B}_i$-run from $\sigma[a_j]$, and vice versa).

We show four claims:

1. For all states $\sigma \in \hat{\Sigma}$ and all $i \in \mathsf{N}$, $\sigma[a_i] \not\models \varphi_{inf}$.

2. For all states $\sigma \in \hat{\Sigma}$ and all $i \in \mathsf{N}$, $\sigma[b_i] \models \varphi_{inf}$.

3. For all TCTL-formulas φ of depth at most i that contain no propositions from $\hat{P}$, for all states $\sigma \in \hat{\Sigma}$ and σ-environments $\mathcal{E}$, and for all integers $j, k \geq i$, $\sigma[a_j] \models_{\mathcal{E}} \varphi$ iff $\sigma[a_k] \models_{\mathcal{E}} \varphi$.

4. For all TCTL-formulas φ of depth at most i that contain no propositions from $\hat{P}$, for all states $\sigma \in \hat{\Sigma}$ and all σ-environments $\mathcal{E}$, $\sigma[a_i] \models_{\mathcal{E}} \varphi$ iff $\sigma[b_i] \models_{\mathcal{E}} \varphi$.

The second part of Theorem 5.1 follows from Claims 1, 2, and 4 applied to $i = \hat{\imath}$; Claim 3 will be used to establish Claim 4.

To show Claim 1, use induction on i.

To show Claim 2, let $s = \{\sigma[b_i] \mid \sigma \in \hat{\Sigma}\}$ and observe that[6]

$$s \;=\; [\![\exists\Diamond(x = 0 \;\wedge\; ((x = 0) \triangleright ((x > 0) \triangleright s)))]\!]_{\Pi_{\mathcal{B}_i}}.$$

The proofs of Claims 3 and 4 are mostly technical and given in the appendix. ∎

Theorem 5.1 should come as no surprise. It is true that in the untimed case, the branching-time logic CTL is strictly less expressive than the propositional μ-calculus [EC80]. More precisely, the until operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$ are definable in the propositional μ-calculus over all systems that are generated by transition relations—i.e., all safe systems [Eme83]. The stutter closure of a safe system, however, may no longer be safe. Indeed, the inevitability operator $\forall\mathcal{U}$ of CTL cannot be defined in the propositional μ-calculus over the stutter closures of safe systems, because the propositional μ-calculus cannot distinguish between a system and its safety closure. Over stutter-closed systems that are safe, the CTL-formula $\varphi_1 \forall\mathcal{U} \varphi_2$ is trivially equivalent to its second argument. As the following proposition shows, this observation applies also in the timed case.

[6] Note that the Tμ-formula φ_{inf} does not define the set s' of states $\sigma \in \Sigma_{\Pi_{\mathcal{B}_i}}$ for which the state predicate $x = 0$ is true at infinitely many positions of some $\mathcal{B}_i$-run that starts from σ (the set s' is empty). Rather, the characteristic set $[\![\varphi_{inf}]\!]_{\Pi_{\mathcal{B}_i}}$ is the set s of states from which $x = 0$ may become true arbitrarily (but finitely) often.

Proposition 5.3 *Let* Π *be a premodel and let* $s_1, s_2 \subseteq \Sigma_\Pi$ *be two sets of reachable states. If* Π *is safe, then* $[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi = s_2$.

Proof. First observe that $s_2 \subseteq [\![s_1 \forall\mathcal{U} s_2]\!]_\Pi$ for all premodels Π.

Now assume that the premodel Π is safe. To see that $[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi \subseteq s_2$, let $\sigma \in \Sigma_\Pi$ be a reachable state and suppose that $\sigma \models_\Pi s_1 \forall\mathcal{U} s_2$. Since Π is future-closed, stutter-closed, and safe, it contains the trajectory

$$\sigma \xrightarrow{0} \sigma \xrightarrow{0} \sigma \xrightarrow{0} \cdots.$$

From $\sigma \models_\Pi s_1 \forall\mathcal{U} s_2$, it follows that $\sigma \in s_2$. ■

In the following subsection, we will show that Tμ is as expressive as TCTL over the class of real-time programs, which will allow us to compute the characteristic sets of TCTL-formulas over real-time programs as fixpoints. Our proof will depend on (1) the divergence safety of real-time programs and (2) the finite variability of TCTL over real-time programs. While our proof of the first part of Theorem 5.1 shows that finite variability by itself is not a sufficient condition for defining inevitability in Tμ, we do not know if divergence safety by itself is sufficient. In other words, it is an open problem if Tμ is as expressive as TCTL over the class of divergence-safe real-time systems.

5.3 Inevitability over real-time programs

We now show in two steps that for the class of real-time programs there is a fixpoint characterization of inevitability. First, recall that real-time programs define divergence-safe real-time systems. Proposition 5.4 will show that over divergence-safe real-time systems, the inevitability operator $\forall\mathcal{U}$ can be defined as a least fixpoint of the time-bounded inevitability operator $\forall\mathcal{U}_{\leq c}$, for any positive integer constant $c > 0$. Second, recall that all TCTL-requirements are finitely variable over the class of real-time programs (Corollary 4.1) and that for all premodels there is a fixpoint characterization of possibility (Proposition 5.1). Lemma 5.1 will show that for finitely variable arguments, the time-bounded inevitability operator $\forall\mathcal{U}_{\leq c}$ is, over the class of real-time systems, definable by the possibility operator $\exists\mathcal{U}$.

Proposition 5.4 *Let* Π *be a real-time system, let* $s_1, s_2 \subseteq \Sigma_\Pi$ *be two sets of reachable states, and let* $c > 0$ *be a positive integer constant. If* Π *is divergence-safe,*[7] *then*

$$[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi \;=\; [\![\mu X.\, (s_2 \;\vee\; (s_1 \forall\mathcal{U}_{\leq c} X))]\!]_\Pi.$$

Proof. First observe that the set $[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi$ is a solution of the equation $X = [\![s_2 \vee (s_1 \forall\mathcal{U}_{\leq c} X)]\!]_\Pi$; that is,

$$[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi \;=\; [\![s_2 \;\vee\; (s_1 \forall\mathcal{U}_{\leq c} (s_1 \forall\mathcal{U} s_2))]\!]_\Pi.$$

Now assume that the real-time system Π is divergence-safe. We show that the set $[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi$ is the least solution of the equation $X = [\![s_2 \vee (s_1 \forall\mathcal{U}_{\leq c} X)]\!]_\Pi$. Let $s \subseteq \Sigma_\Pi$ be such that $s = [\![s_2 \vee (s_1 \forall\mathcal{U}_{\leq c} s)]\!]_\Pi$; we show that $[\![s_1 \forall\mathcal{U} s_2]\!]_\Pi \subseteq s$. Assume that there is a state $\sigma_0 \in ([\![s_1 \forall\mathcal{U} s_2]\!]_\Pi - s)$; we show a contradiction.

We construct inductively an infinite sequence of states $\sigma_i \in ([\![s_1 \forall\mathcal{U} s_2]\!]_\Pi - s)$, for $i \in \mathbb{N}$. Suppose that we have already constructed σ_i. Since $\sigma_i \notin s$, since Π is future-closed, and since all trajectories

[7] We point out that while the condition of divergence safety is not necessary (consider the real-time system that consists of all divergent trajectories along which the value of the proposition p changes finitely often), it cannot be omitted. To see this, recall once again the real-time system Π_2 of Example 2.1, which eventually changes the value of the proposition p from *false* to *true*. From Corollary 5.1 it follows that $[\![\mu X.\, (p \vee \forall\Diamond_{\leq 1} X)]\!]_{\Pi_2} = [\![\mu X.\, (p \vee \forall\Diamond_{\leq 1} X)]\!]_{\overline{\Pi}_2}$, whereas $[\![\forall\Diamond p]\!]_{\Pi_2} \neq [\![\forall\Diamond p]\!]_{\overline{\Pi}_2}$.

in Π diverge, there is a trajectory $\bar{\sigma}_i \in \Pi$ with $\bar{\sigma}(0,0) = \sigma_i$, and a position π_i of $\bar{\sigma}_i$ with $\tau_{\bar{\sigma}}(\pi_i) = c$, such that either

1. for some position $\pi \preceq \pi_i$ of $\bar{\sigma}_i$, $\bar{\sigma}_i(\pi) \notin s_1$ and for all positions $\pi' \preceq \pi$ of $\bar{\sigma}_i$, $\bar{\sigma}_i(\pi') \notin s$; or
2. for all positions $\pi \preceq \pi_i$ of $\bar{\sigma}_i$, $\bar{\sigma}_i(\pi) \in (s_1 - s)$.

But since $\sigma_i \models_\Pi s_1 \forall\mathcal{U} s_2$ and $s_2 \subseteq s$, Condition 1 cannot happen and Condition 2 implies

3. for all positions $\pi \preceq \pi_i$ of $\bar{\sigma}_i$, $\bar{\sigma}_i(\pi) \in ([\![s_1 \forall\mathcal{U} s_2]\!]_\Pi - s)$.

We choose $\sigma_{i+1} = \bar{\sigma}_i(\pi_i)$.

Now consider the trajectory $\bar{\sigma}$ that results from concatening the trajectory prefixes $\bar{\sigma}_i[..\pi_i]$ for all $i \geq 0$:

$$\bar{\sigma} \;=\; \bar{\sigma}_0[..\pi_0] \xrightarrow{0} \bar{\sigma}_1[..\pi_1] \xrightarrow{0} \bar{\sigma}_2[..\pi_2] \xrightarrow{0} \cdots.$$

The trajectory $\bar{\sigma}$ diverges, because $c > 0$. Since Π is fusion-closed and divergence-safe, $\bar{\sigma} \in \Pi$. Furthermore, $\bar{\sigma}(0,0) = \sigma_0$ and, by Condition 3, $\bar{\sigma}(\pi) \notin s$ for all positions π of $\bar{\sigma}$. Since $s_2 \subseteq s$, it follows that $\sigma_0 \not\models_\Pi s_1 \forall\mathcal{U} s_2$, a contradiction. ■

There is a folk theorem (formalized in [Hen92]) that all time-bounded properties are safety requirements. Consider the two time-bounded properties $\forall\Box_{<5}\, \neg q$ and $\forall\Diamond_{\leq 5}\, q$. The time-bounded invariance property $\forall\Box_{<5}\, \neg q$, which puts a lower time bound on the occurrence of a q-state, is written in TCTL as the $\forall\Box$-formula

$$z.\, \forall\Box(z < 5 \;\rightarrow\; \neg q),$$

which is easily recognized as an invariance. On the other hand, the time-bounded response property $\forall\Diamond_{\leq 5}\, q$, which puts an upper time bound on the occurrence of a q-state, is written in TCTL as the $\forall\Diamond$-formula

$$z.\, \forall\Diamond(z \leq 5 \;\wedge\; q).$$

This formula is, over the class of real-time systems, equivalent to the negated possibility (i.e., invariance)

$$\neg z.\, ((\neg q)\exists\mathcal{U}(z > 5)).$$

Intuitively, a q-state is inevitable within time 5 on all divergent trajectories iff it is not possible that more than 5 time units pass without a q-state occurring.

The generalization of this observation to binary bounded-inevitability operators $\forall\mathcal{U}_{\leq c}$ is surprisingly subtle: if the first argument of $\forall\mathcal{U}_{\leq c}$ is different from *true*, we must restrict ourselves to real-time systems over which the second argument is finitely variable.

Lemma 5.1 *Let Π be a real-time system, let $s_1, s_2 \subseteq \Sigma_\Pi$ be two sets of reachable states, and let $c \in \mathbb{N}$ be a nonnegative integer constant. If s_2 is finitely variable over Π, then*

$$[\![s_1 \forall\mathcal{U}_{\leq c}\, s_2]\!]_\Pi \;=\; [\![\neg z.\, ((\neg s_2)\exists\mathcal{U}(\neg(s_1 \,\vee\, s_2) \,\vee\, z > c))]\!]_\Pi.$$

Proof. First, recall that $s_1 \forall\mathcal{U}_{\leq c}\, s_2$ stands for $z.\, (s_1 \forall\mathcal{U}(s_2 \,\wedge\, z \leq c))$, which over all premodels is equivalent to

$$z.\, ((s_1 \,\wedge\, z \leq c)\forall\mathcal{U}(s_2 \,\wedge\, z \leq c)),$$

which over the class of real-time systems is equivalent to

$$z.\, ((s_1 \,\wedge\, z \leq c)\forall\mathbf{U}(s_2 \,\wedge\, z \leq c)),$$

where the *unless* operator $\forall\mathbf{U}$ is defined as follows: for all premodels Π, clock environments $\mathcal{E}$, and reachable states $\sigma \in \Sigma_\Pi$,

$$\sigma \models_{\Pi,\mathcal{E}} \varphi_1 \forall\mathbf{U} \varphi_2$$

iff for all trajectories $\bar{\sigma} \in \Pi$ with $\bar{\sigma}(0,0) = \sigma$, either

1. there exists a position (i,δ) of $\bar{\sigma}$ such that $\bar{\sigma}(i,\delta) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(i,\delta)} \varphi_2$ and for all positions (j,ϵ) of $\bar{\sigma}$, if $(j,\epsilon) \prec (i,\delta)$ then $\bar{\sigma}(j,\epsilon) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(j,\epsilon)} \varphi_1 \vee \varphi_2$; or

2. for all positions (i,δ) of $\bar{\sigma}$, $\bar{\sigma}(i,\delta) \models_{\Pi,\mathcal{E}+\tau_{\bar{\sigma}}(i,\delta)} \varphi_1$.

Second, we show that for all premodels Π and all sets $s_1, s_2 \subseteq \Sigma_\Pi$ of reachable states, if s_2 is finitely variable over Π,[8] then

$$[\![s_1 \forall\mathbf{U} s_2]\!]_\Pi \;=\; [\![\neg((\neg s_2)\exists\mathcal{U}(\neg s_1 \;\wedge\; \neg s_2))]\!]_\Pi.$$

Let $\sigma \in \Sigma_\Pi$ be a reachable state. If $\sigma \models_\Pi (\neg s_2)\exists\mathcal{U}(\neg s_1 \wedge \neg s_2)$, then $\sigma \not\models_\Pi s_1 \forall\mathbf{U} s_2$. Conversely, assume that $\sigma \not\models_\Pi s_1 \forall\mathbf{U} s_2$. Then there is a trajectory $\bar{\sigma} \in \Pi$ with $\bar{\sigma}(0,0) = \sigma$ such that

1. for all positions π of $\bar{\sigma}$ with $\bar{\sigma}(\pi) \in s_2$, there is a position $\pi' \prec \pi$ of $\bar{\sigma}$ such that $\bar{\sigma}(\pi) \notin (s_1 \cup s_2)$; and

2. there is a position π of $\bar{\sigma}$ such that $\bar{\sigma}(\pi) \notin s_1$.

If $\bar{\sigma}(\pi) \notin s_2$ for all positions π of $\bar{\sigma}$, then Condition 2 implies that $\sigma \models_\Pi (\neg s_2)\exists\mathcal{U}(\neg s_1 \wedge \neg s_2)$. Otherwise, let π be the infimum of all positions π' of $\bar{\sigma}$ with $\bar{\sigma}(\pi') \in s_2$; that is, for all positions $\pi' \prec \pi$ of $\bar{\sigma}$, $\bar{\sigma}(\pi') \notin s_2$, and either $\bar{\sigma}(\pi) \in s_2$ or, since s_2 is finitely variable along the trajectory $\bar{\sigma}$, there is a position $\pi' \succ \pi$ of $\bar{\sigma}$ such that for all positions $\pi \prec \pi'' \preceq \pi'$ of $\bar{\sigma}$, $\bar{\sigma}(\pi'') \in s_2$. In either case, Condition 1 implies that $\sigma \models_\Pi (\neg s_2)\exists\mathcal{U}(\neg s_1 \wedge \neg s_2)$. ∎

5.4 Translating timed computation tree logic into the timed μ-calculus

Together, Propositions 5.1 and 5.4, Corollary 4.1, and Lemma 5.1 prescribe a method for computing, over real-time programs, all TCTL-requirements as fixpoints of Tμ. Specifically, we can translate any TCTL-formula φ into a Tμ-formula φ' by

- first replacing each subformula of the form $\psi_1 \forall\mathcal{U} \psi_2$ with the formula

$$\mu X.\,(\psi_2 \;\vee\; \neg z.\,((\neg X)\exists\mathcal{U}(\neg(\psi_1 \vee X) \;\vee\; z > c))),$$

  for any positive integer constant $c > 0$,[9] and

- then replacing each subformula of the form $\psi_1 \exists\mathcal{U} \psi_2$ with the formula $\mu X.\,(\psi_2 \;\vee\; (\psi_1 \triangleright X))$.

The resulting Tμ-formula φ' is equivalent to the original TCTL-formula φ over the class of all real-time programs. We conclude the following theorem.

Theorem 5.2 Tμ *is as expressive as* TCTL *over the class of real-time programs. More specifically, for each* TCTL*-formula φ we can construct a* Tμ*-formula φ' whose size is linear in the size of φ such that φ and φ' are equivalent over the class of real-time programs.*

[8] To see that the finite-variability condition cannot be omitted, consider a real-time system Π that does not reset the clock $x \in C$. Let $s_1 = [\![x = 0]\!]_\Pi$ and $s_2 = \{\sigma \in \Sigma_\Pi \mid \exists n > 0.\, \sigma(x) = \frac{1}{n}\}$. If $\sigma \in \Sigma_\Pi$ with $\sigma(x) = 0$, then $\sigma \not\models_\Pi s_1 \forall\mathbf{U} s_2$ and $\sigma \not\models_\Pi (\neg s_2)\exists\mathcal{U}(\neg s_1 \wedge \neg s_2)$.

[9] The choice of c may effect the number of iterations performed by the symbolic model-checking algorithm of Section 6.1.

6 Symbolic Model Checking

Let φ be a formula of TCTL or Tμ. A premodel Π *meets* (is *correct* with respect to) the requirement φ if all reachable states of Π satisfy the formula φ; that is, if $[\![\varphi]\!]_\Pi = \Sigma_\Pi$. The verification question of deciding if a given premodel, specified in a system description language $\mathcal{A}$, meets a given requirement, specified in a logic $\mathcal{B}$, is called the *model-checking problem* for $\mathcal{A}$ and $\mathcal{B}$.

The model-checking problem for verifying TCTL-specifications φ of real-time programs $\mathcal{P}$ was solved by Alur, Courcoubetis, and Dill [ACD90]. Their solution relies on the explicit construction of a finite quotient graph, called the *region graph*, of the infinite state-transition graph that is defined by the input program $\mathcal{P}$. The construction of the region graph, whose vertices are minimal $(\mathcal{P}, \varphi)$-regions, leads to a model-checking algorithm that is exponential in both the number of input clocks and the size of the largest input constant. The following theorem tells us that we cannot expect more from a general verification procedure.

Theorem 6.1 [ACD90] *The model-checking problem for real-time programs and* TCTL *is PSPACE-complete.*

In practice, however, it is often unnecessary to construct the entire region graph. We take advantage of this observation (1) by representing only those state sets that are required for solving a specific problem instance $(\mathcal{P}, \varphi)$ and (2) by representing state sets symbolically as state predicates. In other words, rather than enumerating all minimal $(\mathcal{P}, \varphi)$-regions, a symbolic model-checking algorithm computes regions selectively and symbolically.

6.1 The algorithm

Let $\mathcal{P}$ be a real-time program and let φ be a formula of TCTL or Tμ. By Theorems 4.1 and 4.2, there is a state predicate ϕ such that $[\![\varphi]\!]_{\Pi_\mathcal{P}} = [\![\phi]\!]$. We call ϕ a *characteristic predicate* of φ for $\mathcal{P}$. We compute characteristic predicates for TCTL-formulas φ in two steps. First, we apply Theorem 5.2 and translate φ into an equivalent Tμ-formula φ'. Second, we apply the following algorithm and compute a characteristic predicate for φ' by successive approximation of fixpoints. The algorithm is defined inductively on the structure of Tμ-formulas.

Algorithm 6.1 (symbolic model checking)

Input: (1) a real-time program $\mathcal{P}$ with the invariant $\phi^\Box$ and
(2) a Tμ-formula φ.

Output: a state predicate $|\varphi|$ such that

- $|p| := \phi^\Box \wedge p;$
- $|x + c \leq y + d| := \phi^\Box \wedge x + c \leq y + d;$
- $|\neg\varphi| := \phi^\Box \wedge \neg|\varphi|;$
- $|\varphi_1 \vee \varphi_2| := |\varphi_1| \vee |\varphi_2|;$
- $|\varphi_1 \triangleright \varphi_2| := |(|\varphi_1| \vee |\varphi_2|) \rightsquigarrow pre_\mathcal{P}(|\varphi_2|)|;$ — see Definition 3.5 for a definition of $pre_\mathcal{P}$ and Theorem 3.2 for the computation of $|\phi_1 \rightsquigarrow \phi_2|$
- $|z.\,\varphi| := |\varphi|[z := 0];$
- $|\mu X.\,\varphi|$ is the result of the following iteration:

ψ := *false*;
repeat
  ϕ := ψ;
  ψ := $|\varphi[X := \phi]|$
until $[\![\phi]\!] = [\![\psi]\!]$;
return ϕ.

■

The test $[\![\phi]\!] = [\![\psi]\!]$, for two state predicates ϕ and ψ, can be decided by using Theorem 3.1 to check the satisfiability of the state predicate $\phi \not\equiv \psi$. The termination and correctness of Algorithm 6.1 follow from the constructive proof of Lemma 4.3.

Theorem 6.2 *Let $\mathcal{P}$ be a real-time program and let φ be a formula of* Tμ. *On input $(\mathcal{P}, \varphi)$, Algorithm 6.1 yields as output, within a finite number of steps, a state predicate $|\varphi|$ such that*

$$[\![|\varphi|]\!] \;=\; [\![\varphi]\!]_{\Pi_{S_\mathcal{P}}}.$$

Moreover, if the real-time program $\mathcal{P}$ is nonzeno, then

$$[\![|\varphi|]\!] \;=\; [\![\varphi]\!]_{\Pi_\mathcal{P}}.$$

Algorithm 6.1 can, therefore, be used to verify TCTL-specifications and Tμ-specifications of nonzeno real-time programs: a nonzeno real-time program $\mathcal{P}$ meets the requirement φ iff the characteristic predicate $|\varphi|$ is equivalent to the program invariant $\phi^\Box$ (use Theorem 3.1 to decide the equivalence of state predicates).

Corollary 6.1 *The model-checking problem for real-time programs and* Tμ *is decidable.*

On one hand, using the region graph of a real-time program, the model checking of Tμ-formulas can be done in time that is exponential in the number of input clocks, the size of the largest input constant, and the nesting depth of fixpoint quantifiers. On the other hand, from the PSPACE-hardness of the reachability problem for timed automata [Alu91], it follows that the model-checking problem for real-time programs and Tμ is PSPACE-hard. The exact complexity, however, is not known to date even for the untimed problem of model checking the propositional μ-calculus.

Many optimizations for computing fixpoints of the propositional μ-calculus apply also to Algorithm 6.1; for example, intermediate results can be reused when nested fixpoints of the same polarity are computed [EL86]. The practicality of a symbolic method depends, furthermore, on the representation of state predicates. While Algorithm 6.1 can be used to compute the characteristic predicates of Tμ-formulas directly on the region graph of the input program, we know of two implementations that represent state sets symbolically and thus avoid the costly construction of the region graph. A verification system at Grenoble [NSY92] represents state predicates by difference matrices over the integers [Dil89]; a verification system at Cornell [AHH93] represents state predicates as Mathematica formulas.

6.2 A verification example

We consider the gate controller of a railroad crossing. The system consists of two parallel processes, namely, a train and a controller. The two processes are modeled as the two timed safety automata shown in Figure 5.

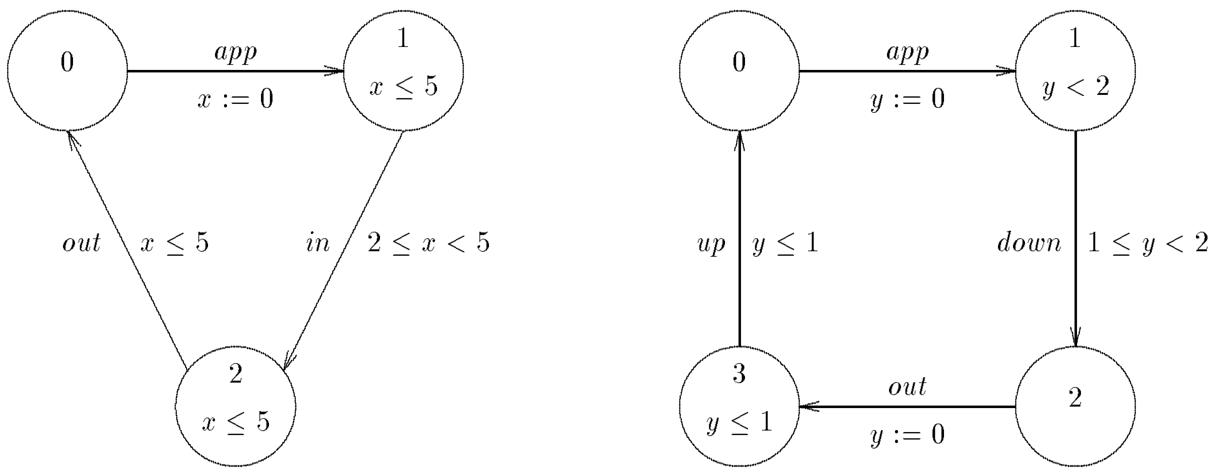

Figure 5: Railroad gate controller

The control variable α of the train automaton ranges over three locations: $\alpha = 0$ if the train is far from the crossing; $\alpha = 1$ if the train is close to the crossing; $\alpha = 2$ if the train is in the crossing. When the train approaches the crossing (i.e., proceeds from location 0 to location 1), it sends the signal *app* to the gate controller. This happens at least 2 minutes before the train enters the crossing. When the train leaves the crossing (i.e., proceeds from location 2 to location 0), it sends the signal *out* to the gate controller. This happens at most 5 minutes after the *app* signal.

The control variable β of the controller automaton ranges over four locations: $\beta = 0$ if the controller is waiting for the train to arrive; $\beta = 1$ if the signal *app* has been received; $\beta = 2$ if the gate is down; $\beta = 3$ if the signal *out* has been received. When the controller receives the *app* signal from the train, it responds by closing the gate within no less than 1 minute and no more than 2 minutes. When the controller receives the *out* signal, it responds by opening the gate within 1 minute.

We can define a product of timed safety automata that combines the train process and the controller process by synchronizing on the *app* and *out* signals. Instead of providing a formal definition of parallel composition (see, for instance, [AD90]), we present a real-time program $\mathcal{P} = (G, \phi^{\Box})$ that specifies the resulting system:

$$
\begin{aligned}
G \; = \{ \; & (\alpha = 0 \wedge \beta = 0) \rightarrow_{app} \alpha := 1, \beta := 1, x := 0, y := 0, \\
& (\alpha = 1 \wedge 2 < x \leq 5) \rightarrow_{in} \alpha := 2, \\
& (\beta = 1 \wedge 1 \leq y < 2) \rightarrow_{down} \beta := 2, \\
& (\alpha = 2 \wedge \beta = 2 \wedge x \leq 5) \rightarrow_{out} \alpha := 0, \beta := 3, y := 0, \\
& (\beta = 3 \wedge y \leq 1) \rightarrow_{up} \beta := 0 \}; \\
\phi^{\Box} \; = \; & \alpha \in \{0,1,2\} \wedge \beta \in \{0,1,2,3\} \wedge \\
& (\alpha \in \{1,2\} \rightarrow x \leq 5) \wedge \\
& (\beta = 1 \rightarrow y < 2) \wedge (\beta = 3 \rightarrow y \leq 1).
\end{aligned}
$$

It is not difficult to check that the real-time program $\mathcal{P}$ is nonzeno.

We wish to verify that the gate is never closed for more than 5 minutes. This requirement is expressed by the TCTL-formula

$$\varphi \;=\; \phi_{init} \;\rightarrow\; \forall\Box(\beta = 2 \;\rightarrow\; \forall\Diamond_{\leq 5}\,\beta = 0),$$

for the initial condition

$$\phi_{init} \;=\; (\alpha = 0 \wedge \beta = 0 \wedge x = 0 \wedge y = 0).$$

By Theorem 5.2, the TCTL-formula φ is over the real-time program $\mathcal{P}$ equivalent to the Tμ-formula

$$\varphi' \;=\; \phi_{init} \;\rightarrow\; \neg\mu Y.\,((\beta = 2 \wedge z.\,\mu X.(z > 5 \vee ((\beta \neq 0) \triangleright X))) \vee (\mathit{true} \triangleright Y)).$$

We now apply Algorithm 6.1 to compute the characteristic predicate $|\varphi'|$.

The state predicate $|\mu X.(z > 5 \vee ((\beta \neq 0) \triangleright X))|$ that defines the inner fixpoint is computed iteratively as $\psi = \bigvee_{i \geq 0} \psi_i$, where $\psi_0 = \mathit{false}$ and

$$\psi_{i+1} \;=\; (\phi^{\Box} \wedge z > 5) \vee ((\phi^{\Box} \wedge \beta \neq 0) \triangleright \psi_i)$$

for all $i \geq 0$. We have:

$$\begin{aligned}
\psi_1 &= (\phi^{\Box} \wedge z > 5);\\
\psi_2 &= \psi_1 \vee ((\alpha = 0 \vee ((\alpha = 1 \vee \alpha = 2) \wedge x \leq 5 \wedge z > x)) \wedge\\
&\quad ((\beta = 1 \wedge y < 2 \wedge z > y + 3) \vee \beta = 2 \vee (\beta = 3 \wedge y \leq 1 \wedge z > y + 4)));\\
\psi_3 &= \psi_2 \vee (\alpha = 0 \wedge \beta = 1 \wedge y < 2) \vee\\
&\quad ((\alpha = 1 \vee \alpha = 2) \wedge \beta = 1 \wedge x \leq 5 \wedge y < 2 \wedge z > x \wedge y \geq x - 4) \vee\\
&\quad (\alpha = 2 \wedge \beta = 2 \wedge x \leq 5 \wedge z > x - 1);\\
\psi_4 &= \psi_3 \vee (x \leq 5 \wedge z > x - 1 \wedge ((\alpha = 1 \wedge \beta = 2) \vee\\
&\quad (\alpha = 2 \wedge \beta = 1 \wedge y < 2 \wedge y \geq x - 4)));\\
\psi_5 &= \psi_4 \vee (\alpha = 1 \wedge \beta = 1 \wedge x \leq 5 \wedge z > x - 1 \wedge y < 2 \wedge y \geq x - 4).
\end{aligned}$$

We find that $[\![\psi_6]\!] = [\![\psi_5]\!]$ and the fixpoint computation terminates with $\psi = \psi_5$. Next we compute the state predicate $|z.\,\psi| = \psi[z := 0]$:

$$\psi' \;=\; (\alpha = 0 \vee ((\alpha = 1 \vee \alpha = 2) \wedge x < 1)) \wedge ((\beta = 1 \wedge y < 2) \vee \beta = 2).$$

Now we compute the outer fixpoint iteratively as $\phi = \bigvee_{i \geq 0} \phi_i$, where $\phi_0 = \mathit{false}$ and

$$\phi_{i+1} \;=\; (\phi^{\Box} \wedge \beta = 2 \wedge \psi') \vee (\phi^{\Box} \triangleright \phi_i)$$

for all $i \geq 0$. This computation converges with $\phi = \phi_2$:

$$\begin{aligned}
\phi_1 &= (\beta = 2 \wedge \psi');\\
\phi_2 &= \phi_1 \vee ((\alpha = 0 \vee ((\alpha = 1 \vee \alpha = 2) \wedge x < 1 \wedge x < y)) \wedge \beta = 1 \wedge y < 2).
\end{aligned}$$

Finally, it is not difficult to check that the the state predicate $(\phi^{\Box} \wedge \phi_{init}) \rightarrow \neg\phi$ is equivalent to the program invariant $\phi^{\Box}$, which implies that the real-time program $\mathcal{P}$ meets the specification φ.

6.3 Symbolic nonzenoness analysis

Algorithm 6.1 can be used (1) to check if a real-time program $\mathcal{P}$ is nonzeno and, if not, (2) to convert $\mathcal{P}$ into an equivalent nonzeno real-time program. To check if $\mathcal{P}$ is nonzeno, we compute the characteristic predicate of the possibility requirement $z.\exists\Diamond(z=1)$, which is true in all states from which time can advance by 1 time unit.

Proposition 6.1 *Let $\mathcal{P}$ be a real-time program with the invariant $\phi^{\Box}$. Then $\mathcal{P}$ is nonzeno iff $[\![z.\exists\Diamond(z=1)]\!]_{\Pi_{S_\mathcal{P}}} = [\![\phi^{\Box}]\!]$.*

Proof. Recall that $[\![\phi^{\Box}]\!] = \Sigma_{S_\mathcal{P}}$ (Proposition 3.1). Therefore $[\![z.\exists\Diamond(z=1)]\!]_{\Pi_{S_\mathcal{P}}} \subseteq [\![\phi^{\Box}]\!]$.

Now consider a state $\sigma_0 \in \Sigma_{S_\mathcal{P}}$ that occurs on an $S_\mathcal{P}$-trajectory. If the real-time program $\mathcal{P}$ is nonzeno, then every state that occurs on an $S_\mathcal{P}$-trajectory occurs also on a run of $\mathcal{P}$ and, therefore, $\sigma_0 \models_{\Pi_{S_\mathcal{P}}} z.\exists\Diamond(z=1)$. Conversely, if all states $\sigma \in \Sigma_{S_\mathcal{P}}$ that occur on $S_\mathcal{P}$-trajectories satisfy $\sigma \models_{\Pi_{S_\mathcal{P}}} z.\exists\Diamond(z=1)$, then we can inductively construct a divergent $S_\mathcal{P}$-trajectory that starts from σ_0 (for all $i \in \mathsf{N}$, add a segment whose length is exactly 1 time unit from σ_i to a state σ_{i+1}). ■

To convert a zeno real-time program $\mathcal{P}$ into an equivalent nonzeno program, we need to strengthen the program invariant so that the new invariant rules out all states from which time cannot diverge. The "bad" (zeno) states of $\Pi_{S_\mathcal{P}}$ are the states that do not satisfy the Tμ-formula[10]

$$\varphi_{nz} \;=\; \nu X.\, z.\exists\Diamond(z=1 \wedge X),$$

which is true in all states that occur on an $S_\mathcal{P}$-trajectory along which time advances infinitely often by 1 time unit; that is, φ_{nz} characterizes precisely the states that occur on runs of $\mathcal{P}$.

Proposition 6.2 *For every real-time program $\mathcal{P}$, $[\![\varphi_{nz}]\!]_{\Pi_{S_\mathcal{P}}} = \Sigma_\mathcal{P}$.*

The characteristic predicate of the greatest-fixpoint formula φ_{nz} can be computed with Algorithm 6.1. Since the invariant $\phi^{\Box}$ of any nonzeno real-time program $\mathcal{P}$ defines precisely the set $\Sigma_\mathcal{P}$ of states that occur on runs of $\mathcal{P}$ (Proposition 3.1), the invariant $\phi^{\Box}$ is equivalent to the characteristic predicate $|\varphi_{nz}|$ (Proposition 6.2). On the other hand, if the real-time program $\mathcal{P}$ is zeno, then the invariant $\phi^{\Box}$ is weaker than $|\varphi_{nz}|$ and can be replaced by $|\varphi_{nz}|$ to obtain an equivalent nonzeno program.

Theorem 6.3 *For every real-time program $\mathcal{P}$ there is an equivalent nonzeno real-time program, which can be obtained from $\mathcal{P}$ by replacing the program invariant with the state predicate $|\varphi_{nz}|$ (as computed by Algorithm 6.1).*

6.4 Concluding remarks

We discussed two applications of symbolic model checking—the automatic conversion of a real-time program into an equivalent nonzeno program, and the verification of nonzeno real-time programs.

The ideas that underly our method can be used to devise various different symbolic verification strategies. Suppose that we wish to check for the real-time program $\mathcal{P}$ that no ϕ_2-state can be reached from a ϕ_1-state (ϕ_1 might define the initial states of $\mathcal{P}$ and ϕ_2 some set of "bad"

[10] The Tμ-formula φ_{nz} is not equivalent to any TCTL-formula over all premodels. However, one can define an extension TCTL* of TCTL, analogous to the extension CTL* [EH86] of CTL, which separates the trajectory quantifiers (e.g., $\forall$) from the temporal operators (e.g., $\Box$). In TCTL*, the formula φ_{nz} can be written as $\exists\Box z.\Diamond(z=1)$.

states). We solve this problem "backwards": starting from the set of ϕ_2-states, we iterate the precondition operator $pre_{\mathcal{P}}$ (program transitions) and the evolves-to operator $\rightsquigarrow$ (time delays) to compute the set $[\![\exists\Diamond\phi_2]\!]_{\Pi_{\mathcal{P}}}$ of states that reach a ϕ_2-state; then we check if this set contains a ϕ_1-state. Alternatively, one can define a symbolic postcondition operator $post_{\mathcal{P}}$ and a symbolic evolves-from operator $\rightsquigarrow^{-1}$ to solve the reachability problem "forwards": starting from the set of ϕ_1-states, iterate $post_{\mathcal{P}}$ and $\rightsquigarrow^{-1}$ to compute the set s of states that are reachable from a ϕ_1-state; then check if s contains a ϕ_2-state. A third strategy constructs, also by successive approximation, the coarsest bisimulation of the state space that separates all ϕ_1-states from ϕ_2-states. This strategy, too, can be implemented using the precondition and evolves-to operators [ACH+92].

We conclude by pointing out that a natural extension of our method allows the analysis of so-called *hybrid systems*, which contain variables that change continuously in more general ways than clocks [ACHH93, NOSY93].

Acknowledgment. We thank Peter Kopke for a careful reading.

Appendix

Completion of the proof of Theorem 5.1. To show Claim 3, we define an equivalence relation on pairs consisting of states and clock environments. Let $(\sigma_1, \mathcal{E}_1) \sim (\sigma_2, \mathcal{E}_2)$ iff for all state predicates ϕ, $\sigma_1[\mathcal{E}_1] \models \phi$ iff $\sigma_2[\mathcal{E}_2] \models \phi$; that is, $(\sigma_1, \mathcal{E}_1)$ and $(\sigma_2, \mathcal{E}_2)$ differ at most in the fractional parts of clock values and the relative order of all fractional parts of clock values is preserved. For example, for five clocks x, y, z, u, and v,

$$[x := 1, y := 1.1, z := 1.9, u := 3.5, v := 7.9] \ \sim\ [x := 1, y := 1.6, z := 1.8, u := 3.7, v := 7.8].$$

Let $\mathcal{P}$ be a real-time program and assume that $(\sigma_1, \mathcal{E}_1) \sim (\sigma_2, \mathcal{E}_2)$. Then $\sigma_1[\mathcal{E}_1]$ and $\sigma_2[\mathcal{E}_2]$ belong to the same minimal $(\mathcal{P}, \varphi)$-region, for any TCTL-formula φ. From Theorem 4.1 it follows that for all TCTL-formulas φ, $\sigma_1[\mathcal{E}_1] \models_{\Pi_{\mathcal{P}}} \phi$ iff $\sigma_2[\mathcal{E}_2] \models_{\Pi_{\mathcal{P}}} \phi$; that is, $\sigma_1 \models_{\Pi_{\mathcal{P}}, \mathcal{E}_1} \phi$ iff $\sigma_2 \models_{\Pi_{\mathcal{P}}, \mathcal{E}_2} \phi$. We call this observation the *equivalent-values lemma.*

Now let us turn to Claim 3. It suffices to show by induction on the structure of φ that for all $j \geq i$, $\sigma[a_{j+1}] \models_{\mathcal{E}} \varphi$ iff $\sigma[a_j] \models_{\mathcal{E}} \varphi$. The case that φ is atomic ($i = 0$) depends on the assumption that φ contains no propositions from $\hat{P}$. The cases that φ is of the form $\neg\psi$, $\psi_1 \vee \psi_2$, or $z.\psi$ are straightforward. So assume that $\sigma[a_{j+1}] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; that is, there exist two delays $\delta_{j+1}, \delta_j > 0$ such that $\sigma(y) + \delta_{j+1} + \delta_j < 1$ and

- $\sigma[a_{j+1}] + \delta_{j+1} \models_{\mathcal{E}+\delta_{j+1}} \psi_2$ and
  for all $\epsilon < \delta_{j+1}$, $\sigma[a_{j+1}] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or
- $(\sigma[a_j] + \delta_{j+1})[x := 0] + \delta_j \models_{\mathcal{E}+\delta_{j+1}+\delta_j} \psi_2$, and
  for all $\epsilon < \delta_j$, $(\sigma[a_j] + \delta_{j+1})[x := 0] + \epsilon \models_{\mathcal{E}+\delta_{j+1}+\epsilon} \psi_1 \vee \psi_2$, and
  for all $\epsilon \leq \delta_{j+1}$, $\sigma[a_{j+1}] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or
- $(\sigma[a_{j-1}] + \delta_{j+1} + \delta_j)[x := 0] \models_{\mathcal{E}+\delta_{j+1}+\delta_j} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $\epsilon \leq \delta_j$, $(\sigma[a_j] + \delta_{j+1})[x := 0] + \epsilon \models_{\mathcal{E}+\delta_{j+1}+\epsilon} \psi_1 \vee \psi_2$, and
  for all $\epsilon \leq \delta_{j+1}$, $\sigma[a_{j+1}] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$.

Observe that $\delta_{j+1} > 0$ implies

$$((\sigma[a_{j-1}] + \delta_{j+1} + \delta_j)[x := 0], \mathcal{E} + \delta_{j+1} + \delta_j) \ \sim\ ((\sigma[a_{j-1}] + \delta_{j+1})[x := 0], \mathcal{E} + \delta_{j+1}).$$

Using the induction hypothesis and, in the last case, the equivalent-values lemma, we conclude that $\sigma[a_j] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; namely,

- $\sigma[a_j] + \delta_{j+1} \models_{\mathcal{E}+\delta_{j+1}} \psi_2$ and
  for all $\epsilon < \delta_{j+1}$, $\sigma[a_j] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{j-1}] + \delta_{j+1})[x := 0] + \delta_j \models_{\mathcal{E}+\delta_{j+1}+\delta_j} \psi_2$, and
  for all $\epsilon < \delta_j$, $(\sigma[a_{j-1}] + \delta_{j+1})[x := 0] + \epsilon \models_{\mathcal{E}+\delta_{j+1}+\epsilon} \psi_1 \vee \psi_2$, and
  for all $\epsilon \leq \delta_{j+1}$, $\sigma[a_j] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{j-1}] + \delta_{j+1})[x := 0] \models_{\mathcal{E}+\delta_{j+1}} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $\epsilon \leq \delta_{j+1}$, $\sigma[a_j] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$.

The converse implication and the case that φ is of the form $\psi_1 \forall\mathcal{U} \psi_2$ are checked similarly.

To show Claim 4, we use again induction on the structure of φ. The atomic, boolean, and reset-quantifier cases are easily checked. So assume that $\sigma[a_i] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; that is, there exists a delay $\delta > 0$ such that $\sigma(y) + \delta < 1$ and

- $\sigma[a_i] + \delta \models_{\mathcal{E}+\delta} \psi_2$ and
  for all $\epsilon < \delta$, $\sigma[a_i] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{i-1}] + \delta)[x := 0] \models_{\mathcal{E}+\delta} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $\epsilon \leq \delta$, $\sigma[a_i] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$.

Using the induction hypothesis, we conclude that $\sigma[b_i] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; namely,

- $\sigma[b_i] + \delta \models_{\mathcal{E}+\delta} \psi_2$ and
  for all $\epsilon < \delta$, $\sigma[b_i] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{i-1}] + \delta)[x := 0] \models_{\mathcal{E}+\delta} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $\epsilon \leq \delta$, $\sigma[b_i] + \epsilon \models_{\mathcal{E}+\epsilon} \psi_1 \vee \psi_2$.

Conversely, assume that $\sigma[b_i] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; that is, there exists a sequence $\delta_0, \ldots, \delta_n > 0$ of delays such that $\sigma(y) + \Delta_n < 1$, for $\Delta_j = \sum_{0 \leq k \leq j} \delta_k$, and

- $(\sigma[b_i] + \Delta_{n-1})[x := 0] + \delta_n \models_{\mathcal{E}+\delta_n} \psi_2$ and
  for all $\epsilon < \delta_n$, $(\sigma[b_i] + \Delta_{n-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{n-1}+\epsilon} \psi_1 \vee \psi_2$, and
  for all $j < n$ and all $\epsilon \leq \delta_j$, $(\sigma[b_i] + \Delta_{j-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{j-1}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{i-1}] + \Delta_n)[x := 0] \models_{\mathcal{E}+\Delta_n} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $j \leq n$ and all $\epsilon \leq \delta_j$, $(\sigma[b_i] + \Delta_{j-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{j-1}+\epsilon} \psi_1 \vee \psi_2$.

Using the induction hypothesis and Claim 3, we conclude that $\sigma[a_{i+n}] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$; namely,

- $(\sigma[a_i] + \Delta_{n-1})[x := 0] + \delta_n \models_{\mathcal{E}+\delta_n} \psi_2$ and
  for all $\epsilon < \delta_n$, $(\sigma[a_i] + \Delta_{n-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{n-1}+\epsilon} \psi_1 \vee \psi_2$, and
  for all $j < n$ and all $\epsilon \leq \delta_j$, $(\sigma[a_{i+n-j}] + \Delta_{j-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{j-1}+\epsilon} \psi_1 \vee \psi_2$, or

- $(\sigma[a_{i-1}] + \Delta_n)[x := 0] \models_{\mathcal{E}+\Delta_n} \psi_1 \exists\mathcal{U} \psi_2$, and
  for all $j \leq n$ and all $\epsilon \leq \delta_j$, $(\sigma[a_{i+n-j}] + \Delta_{j-1})[x := 0] + \epsilon \models_{\mathcal{E}+\Delta_{j-1}+\epsilon} \psi_1 \vee \psi_2$.

Another application of Claim 3 yields $\sigma[a_i] \models_{\mathcal{E}} \psi_1 \exists\mathcal{U} \psi_2$. The case that φ is of the form $\psi_1 \forall\mathcal{U} \psi_2$ is checked similarly. ■

References

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D.L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In S.A. Smolka, editor, *CONCUR 92: Theories of Concurrency*, Lecture Notes in Computer Science 630, pages 340–354. Springer-Verlag, 1992.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In A. Nerode, editor, *Theory of Hybrid Systems*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In M.S. Paterson, editor, *ICALP 90: Automata, Languages, and Programming*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, 1990.

[ADS86] B. Alpern, A.J. Demers, and F.B. Schneider. Safety without stuttering. *Information Processing Letters*, 23(4):177–180, 1986.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.

[AH92a] R. Alur and T.A. Henzinger. Back to the future: towards a theory of timed regular languages. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society Press, 1992.

[AH92b] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 74–106. Springer-Verlag, 1992.

[AHH93] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual Real-time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 165–175. IEEE Computer Society Press, 1988.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 1–27. Springer-Verlag, 1992.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[CBM89] O. Coudert, C. Berthet, and J. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *CAV 89: Automatic Verification Methods for Finite-state Systems*, Lecture Notes in Computer Science 407, pages 365–373. Springer-Verlag, 1989.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *CAV 89: Automatic Verification Methods for Finite-state Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer-Verlag, 1989.

[EC80] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs as fixpoints. In *ICALP 83: Automata, Languages, and Programming*, Lecture Notes in Computer Science 85, pages 169–181. Springer-Verlag, 1980.

[EC82] E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[EH86] E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not never" revisited: on branching versus linear-time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EL86] E.A. Emerson and C. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, pages 267–278. IEEE Computer Society Press, 1986.

[Eme83] E.A. Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26(1):121–130, 1983.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers (North-Holland), 1990.

[Eme92] E.A. Emerson. Real time and the μ-calculus. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 176–194. Springer-Verlag, 1992.

[EMSS90] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In R.P. Kurshan and E.M. Clarke, editors, *CAV 90: Computer-aided Verification*, Lecture Notes in Computer Science 531, pages 136–145. Springer-Verlag, 1990.

[FR75] J. Ferrante and C. Rackoff. A decision procedure for the first-order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[Hen92] T.A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[HLW91] U. Holmer, K. Larsen, and Y. Wang. Deciding properties of regular real timed processes. In K. Larsen and A. Skou, editors, *CAV 91: Computer-aided Verification*, Lecture Notes in Computer Science 575, pages 443–453. Springer-Verlag, 1991.

[HMP91] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th Annual Symposium on Principles of Programming Languages*, pages 353–366. ACM Press, 1991.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[Lew90] H.R. Lewis. A logic of concrete time intervals. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 380–389. IEEE Computer Society Press, 1990.

[McM93] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In A. Nerode, editor, *Theory of Hybrid Systems*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K. Larsen and A. Skou, editors, *CAV 91: Computer-aided Verification*, Lecture Notes in Computer Science 575, pages 376–398. Springer-Verlag, 1991.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, SE-18(9):794–804, 1992.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.

[QS81] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Fifth International Symposium on Programming*, Lecture Notes in Computer Science 137, pages 337–351. Springer-Verlag, 1981.

[Sif82] J. Sifakis. A unified approach for studying the properties of transition systems. *Theoretical Computer Science*, 18:227–258, 1982.