Verification Methods for the Divergent Runs of Clock Systems*†

Thomas A. Henzinger‡ and Peter W. Kopke§

Computer Science Department
Cornell University, Ithaca, NY 14853

Abstract. We present a methodology for proving temporal properties of the divergent runs of reactive systems with real-valued clocks. A run *diverges* if time advances beyond any bound. Since the divergent runs of a system may satisfy liveness properties that are not satisfied by some convergent runs, the standard proof rules are incomplete if only divergent runs are considered.

First, we develop a sound and complete proof calculus for divergence, which is based on translating clock systems into discrete systems. Then, we show that simpler proofs can be obtained for stronger divergence assumptions, such as *unknown ϵ-divergence*, which requires that all delays have a minimum duration of some unknown constant ϵ. We classify all real-time systems into an infinite hierarchy, according to how well they admit the translation of eventuality properties into equivalent safety properties.

1 Introduction

Several researchers have devised sound and complete calculi for proving temporal properties of concurrent systems [MP89, Schar]. The introduction of real-valued time into systems provides new difficulties to be overcome. A standard model for real-time systems distinguishes between system transitions, which are instantaneous, and delay steps, during which time passes [HMP91, AL92, LV92, AH93]. This model admits deviant runs—those in which time converges to a finite limit. We call the nondeviant runs *divergent* and study the verification of temporal properties for the divergent runs of real-time systems.

We model the state of a real-time system by discrete data variables and real-valued clocks [AD90, AH93]. Consider, for example, the clock program *Wait5* of Figure 1. The variable c is a clock that measures the elapsed time. The instruction $c < 5 \rightarrow$ **wait** allows some amount of time to pass, but not so much that in the resulting state $c > 5$. As long as time is allowed to progress for

*A preliminary version of this paper appeared in the *Proceedings of the Third International Symposium on Formal Techniques in Real-time and Fault-tolerant Systems* (FTRTFT 94), *Lecture Notes in Computer Science* **863**, Springer-Verlag, 1994, pp. 351–372.

†Supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

‡E-mail address: tah@cs.cornell.edu

§Supported in part by the U.S. Army Research Office through the Mathematical Sciences Institute of Cornell University, Contract Number DAAL03-91-C-0027. E-mail address: pkpk@cs.cornell.edu

Program *Wait5*:
$x : \mathbb{N}$;
c : Clock;
initially $x = 1 \wedge c = 0$;
ℓ_1: **while** $c < 5$ **do**
ℓ_2: $c < 5 \rightarrow$ **wait**
od;
ℓ_3: $x := 0$.

Figure 1: A simple clock program

at least 5 units, a state in which $x = 0$ will be reached. So the program *Wait5* satisfies $\Diamond(x = 0)$ over all divergent runs. Now consider a run in which less than 5 time units pass. For example, if the first iteration takes 1 time unit, the second takes $\frac{1}{2}$, the third takes $\frac{1}{4}$, etc., then control never leaves the while loop. Such a run does not satisfy $\Diamond(x = 0)$.

While there has been previous research on the definition of liveness and proof calculi for real-time systems [HMP91, AL92, GSSAL93, MP93a], we know of no systematic and complete approach for proving properties of divergent runs only. On the contrary, there are several obvious, but ultimately unsatisfactory, attempts at proving properties for divergent runs. For example, one might hope to provide a fairness-like rule, using helpful transitions to ensure divergence. This does not seem to be possible in general, for if all waiting steps are helpful, then time can converge via delays of duration $1, \frac{1}{2}, \frac{1}{4}, \ldots$ On the other hand, if only some delays are helpful, then some divergent runs will be excluded.

Alternatively, one may observe that the temporal formula $(\forall n)\Diamond(now > n)$ specifies divergence, where *now* is a flexible variable that measures the elapsed time and n is a rigid variable. Then one might attempt to prove that a property ϕ holds of the divergent runs by proving that the temporal formula

$$((\forall n)\Diamond(now > n)) \Rightarrow \phi$$

holds of all runs. This is equivalent to proving for all runs an existential temporal formula of the form $(\exists n)\varphi(n)$, where the witness for n may depend on the chosen run. We do not know of a general calculus for proving such a property.

We provide two approaches for proving $R \models_{div} \phi$—that all divergent runs of the real-time system R satisfy the temporal property ϕ. The first, *system-transforming*, approach (Section 3) reduces the problem $R \models_{div} \phi$ to the problem of proving the property ϕ for certain runs of an untimed system $\overline{R}$. The second, *property-transforming*, approach (Section 4) reduces the problem $R \models_{div} \phi$ to the problem of proving a safety property ϕ_S for all runs of R. We now briefly preview both approaches.

Characterizing divergence through system transformation. In Section 3, we provide a sound and complete calculus for proving temporal properties of the divergent runs of a guarded-command program with clock variables. Given a clock program R, we generate a discrete system $\overline{R}$, called the region system [AD90], such that (1) the runs of $\overline{R}$ satisfy the same dense-time temporal formulas as the runs of R, and (2) divergence is definable by a discrete-time temporal formula ψ. To prove that the property ϕ holds of all divergent runs of R, we then appeal to an untimed calculus to provide a proof that $\psi \Rightarrow \phi$ holds of all runs of $\overline{R}$.

Appealing to divergence for property transformation. While the system-transforming proof method is theoretically appealing (i.e., sound and complete), the formula ψ expressing divergence is a response ($\Box\Diamond$) property, and so a proof via the region system always requires complex temporal reasoning. Therefore it is desirable to avoid the region system when proving safety ($\Box$) and eventuality ($\Diamond$) properties. There is a natural condition (nonzenoness) on timed systems that allows the proof of safety properties in the usual way, with invariants [AL92, Hen92]. In Section 4, we transform eventualities into equivalent safety properties. This can be achieved only if we strengthen the concept of divergence.

Inspired by [AAT94, AH94], we introduce *unknown ϵ-divergence*. In this model, each delay has at least duration ϵ, for an unknown but fixed value of ϵ. Unknown ϵ-divergence may admit the transformation of eventualities into safety properties, taking ϵ as a parameter. We classify real-time systems as to how well they admit this translation.

2 Definitions

We briefly review the clock model for real-time systems, as presented in [AH93].[1]

We assume a fixed set V of typed *data variables*, together with a set of terms and atomic formulas over V, and a fixed set C of *clock variables*, which we call *clocks*. The type of each clock is the set $\mathbb{R}^+$ of nonnegative real numbers. A *state* is a function σ that maps each variable in $V \cup C$ to a value of the appropriate type. The function σ is extended to terms over V in the standard way.

A *data predicate* is a formula of predicate logic whose free variables are elements of V. A *clock constraint* is an atomic formula of the form $c \leq n$, $c \geq n$, $c \leq d+n$, or $c \geq d+n$, where c and d are clocks, and n is a natural number. A *state predicate* is a boolean combination of data predicates and clock constraints. The state σ *satisfies* the state predicate φ, written $\sigma \models \varphi$, if the statement formed from φ by replacing each free variable x by $\sigma(x)$, is true.

Let V' be the "primed" version of V, i.e., $V' = \{x' \mid x \in V\}$. An *action predicate* is a formula of predicate logic whose free variables are elements of $V \cup V'$. The pair (σ, τ) of states *satisfies* the action predicate ψ, written $(\sigma, \tau) \models \psi$, if the statement formed from ψ by replacing each free variable x from V by $\sigma(x)$, and replacing each free variable x' from V' by $\tau(x)$, is true.

An *untimed system* S is a pair $(\varphi_0, \psi_\triangleright)$, where φ_0 is a data predicate called the *initial condition*, and $\psi_\triangleright$ is an action predicate called the *transition condition*. A *run* of S is an infinite sequence (σ_n) of states such that $\sigma_0 \models \varphi_0$, and for each $n \in \mathbb{N}$, either $(\sigma_n, \sigma_{n+1}) \models \psi_\triangleright$ or $\sigma_n = \sigma_{n+1}$.

2.1 Real-time system description language: guarded commands and guarded delays

A *guarded command* ψ is a pair (φ, f), where φ is a state predicate and f is a map that takes each data variable to a term over V, and each clock c either to c or to 0. The guarded command ψ is *enabled* in the state σ if $\sigma \models \varphi$. The execution of ψ results in the state $f(\sigma)$, defined by $f(\sigma)(x) = \sigma(f(x))$ for each variable $x \in V \cup C$. Notice that for every clock c, the execution of a guarded command either leaves the value of c unchanged, or resets the value of c to 0.

A *guarded delay* is a state predicate. Given a state σ, a guarded delay χ, and a nonnegative real t, we say σ *admits* χ *for duration* t if for each nonnegative real s with $s < t$, $\sigma + s \models \chi$, where $\sigma + s$ is the state τ such that τ agrees with σ on all data variables, and $\tau(c) = \sigma(c) + s$ for each clock c. The execution of χ for duration t results in the state $\sigma + t$.

[1] The clock model originated with timed automata [AD90] and timed safety automata [HNSY92].

A *clock system* R is a triple (φ_0, Ψ, X), where φ_0 is a state predicate called the *initial condition*, Ψ is a finite set of guarded commands called the *transition condition*, and X is a finite nonempty set of guarded delays called the *environment condition*. For convenience, we assume that φ_0 implies $c = 0$ for every clock c. We define two transition relations on states, the first corresponding to the execution of a guarded command from Ψ, and the second corresponding to the execution of a guarded delay from X.

Command successor relation $\triangleright_\Psi$. For each state σ and each guarded command $(\varphi, f) \in \Psi$, if (φ, f) is enabled in σ, then $\sigma \triangleright_\Psi f(\sigma)$.

Delay successor relation $\triangleright_X$. For each state σ, each guarded delay $\chi \in X$, and each nonnegative real t, if σ admits χ for duration t, then $\sigma \triangleright_X^t \sigma + t$. Notice that $\sigma \triangleright_X^0 \sigma$ and define $\triangleright_X = \bigcup_{t \in \mathbb{R}^+} \triangleright_X^t$.

A *run* ρ of R is an infinite sequence (σ_n) of states such that $\sigma_0 \models \varphi_0$, and for each $n \in \mathbb{N}$, either $\sigma_n \triangleright_\Psi \sigma_{n+1}$ or $\sigma_n \triangleright_X \sigma_{n+1}$. The *elapsed time* $T_\rho(n)$ at the state σ_n of ρ is defined inductively: $T_\rho(0) = 0$, and

$$T_\rho(n+1) = \begin{cases} T_\rho(n) & \text{if } \sigma_n \triangleright_\Psi \sigma_{n+1}, \\ T_\rho(n) + t & \text{if } \sigma_n \triangleright_X^t \sigma_{n+1}. \end{cases}$$

The run ρ is *divergent* if $lim_n T_\rho(n) = \infty$.[2]

The clock system R is *nonzeno* if every state that occurs on some run of R, also occurs on a divergent run of R.[3] Nonzeno clock systems are executable, because the stepwise execution of a nonzeno system can never paint itself into a corner from which time cannot diverge. Given a clock system R with finitely many data states[4] (a so-called *timed safety automaton*), it is possible to automatically generate a nonzeno system with the same divergent runs as R [HNSY92]. In general, the problem of checking if a clock system is nonzeno is, of course, undecidable. In Section 3, we present a proof method for establishing the nonzenoness of clock systems.

2.2 Requirement specification language: temporal logic

Let R be a clock system. Traditional (discrete-time) temporal logics are inadequate for specifying the behavior of R, because they do not take into account the (uncountably many) states that are passed by delays [AH92]. For example, if *now* is a clock that measures the elapsed time, then every divergent run ρ of R should satisfy the property $\Diamond(now = 1)$, which asserts that there is a state at elapsed time 1. Yet if the first action of ρ is a delay of duration 2, then ρ does not satisfy $\Diamond(now = 1)$ under the usual definition of temporal satisfaction. To avoid this problem, we use a dense-time temporal logic for specifying properties of R. The semantics we use are called *super-dense* in [MP93b].

Given a run $\rho = (\sigma_n)$ of R, we define the set of *positions*

$$P_\rho = \{(n, \delta) \in \mathbb{N} \times \mathbb{R}^+ \mid \delta \leq T_\rho(n+1) - T_\rho(n)\}.$$

We write $\leq$ for the lexicographic order on P_ρ; this order corresponds to the temporal order of positions along ρ. The *state at position* (n, δ) *of* ρ, denoted $\rho(n, \delta)$, is $\sigma_n + \delta$. The *elapsed time at position* (n, δ) *of* ρ, denoted $T_\rho(n, \delta)$, is $T_\rho(n) + \delta$.

[2] A divergent run is called *admissible* in [LV92].

[3] A nonzeno system is termed *feasible* in [LV92].

[4] A *data state* is an interpretation for the data variables only.

Our temporal logic is based on the *until* operator $\mathcal{U}$ and the *since* operator $\mathcal{S}$. The *temporal formulas* are generated by the grammar

$$\phi ::= \varphi \mid \mathit{first} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2$$

where φ is a state predicate. Abbreviations are defined in the usual way; for example, $\Diamond\phi = \mathit{true}\mathcal{U}\phi$ ("eventually ϕ"), $\Box\phi = \neg\Diamond\neg\phi$ ("always ϕ"), and $\Diamondminus\phi = \mathit{true}\mathcal{S}\phi$ ("sometime in the past ϕ"). A *past formula* is a temporal formula that does not contain the until operator. A $\Box$ *formula* (resp. $\Diamond$ *formula*) is a formula of the form $\Box q$ (resp. $\Diamond q$), where q is a past formula.

The satisfaction relation $\models$ for the run ρ, the position $(n, \delta) \in P_\rho$, and the temporal formula ϕ, is defined by induction on ϕ:

$$\begin{array}{lll}
(\rho,(n,\delta)) \models \varphi & \text{iff} & \rho(n,\delta) \models \varphi; \\
(\rho,(n,\delta)) \models \mathit{first} & \text{iff} & (n,\delta) = (0,0); \\
(\rho,(n,\delta)) \models \neg\phi & \text{iff} & (\rho,(n,\delta)) \not\models \phi; \\
(\rho,(n,\delta)) \models \phi_1 \vee \phi_2 & \text{iff} & (\rho,(n,\delta)) \models \phi_1 \text{ or } (\rho,(n,\delta)) \models \phi_2; \\
(\rho,(n,\delta)) \models \phi_1 \mathcal{U} \phi_2 & \text{iff} & \text{there is a position } (l,\beta) \geq (n,\delta) \text{ such that} \\
 & & (\rho,(l,\beta)) \models \phi_2, \text{ and for each position } (m,\gamma) \text{ with} \\
 & & (n,\delta) \leq (m,\gamma) < (l,\beta),\ (\rho,(m,\gamma)) \models \phi_1 \vee \phi_2; \\
(\rho,(n,\delta)) \models \phi_1 \mathcal{S} \phi_2 & \text{iff} & \text{there is a position } (l,\beta) \leq (n,\delta) \text{ such that} \\
 & & (\rho,(l,\beta)) \models \phi_2, \text{ and for each position } (m,\gamma) \text{ with} \\
 & & (l,\beta) < (m,\gamma) \leq (n,\delta),\ (\rho,(m,\gamma)) \models \phi_1 \vee \phi_2.
\end{array}$$

Notice the slight change in the definition of the satisfaction relation for the until and since operators from the usual. This is required in the dense-time setting, to make sure that, for example, if a clock c starts at 0 and encounters a delay of duration 2, then the formula $(c \leq 1)\mathcal{U}(c > 1)$ is satisfied (this would fail if we used the stronger requirement "$(\rho,(m,\gamma)) \models \phi_1$" in the definition of satisfaction for until formulas) [HNSY92].

The run ρ *satisfies* the temporal formula ϕ, denoted $\rho \models \phi$, if $(\rho,(0,0)) \models \phi$. The clock system R *satisfies* ϕ, written $R \models_{div} \phi$, if every divergent run of R satisfies ϕ.

We also interpret temporal formulas over the runs of untimed systems. In this case, the set of positions of a run is $\mathbb{N} \times \{0\}$, and our definition of the satisfaction relation coincides with the usual one [MP92]. When dealing with untimed systems, we freely use *next* and *previous* operators [MP92, Schar]: $\bigcirc$ inspects the value of a term or formula in the next state of a run; $\ominus$ inspects the value of a term or formula in the previous state, and is defined arbitrarily in the first state of a run, except on boolean expressions where it is false.

The untimed system S *satisfies* the temporal formula ϕ, written $S \models \phi$, if every run of S satisfies ϕ.

2.3 Specifying timing requirements

Since the temporal operators contain no explicit references to time, the only way our logic can express timing requirements of a system is by referring to the clocks of the system. Therefore it is convenient to augment each clock system with a clock *now* that measures the elapsed time.

Let $R = (\varphi_0, \Psi, X)$ be a clock system. Let $\mathit{now} \notin C$ be a new clock. Define $\varphi_{now} = (\varphi_0 \wedge \mathit{now} = 0)$. For each guarded command $(\varphi, f) \in \Psi$, define $(\varphi, f)_{now} = (\varphi, f \cup \{\mathit{now} \mapsto \mathit{now}\})$;[5] that is, $(\varphi, f)_{now}$ acts as (φ, f) on $V \cup C$, and leaves *now* unchanged. Define $\Psi_{now} = \{(\varphi, f)_{now} \mid (\varphi, f) \in$

[5] By $x \mapsto f(x)$ we indicate that the function f maps the variable x to $f(x)$.

$\Psi\}$. The clock system $R_{now} = (\varphi_{now}, \Psi_{now}, X)$ over the set V of data variables and the set $C \cup \{now\}$ of clocks, is called the *elapsed-time system* for R. There is a natural one-to-one correspondence between the runs of R and the runs of R_{now}: each state σ_n on a run $\rho = (\sigma_n)$ of R corresponds to the augmented state $\sigma_n \cup \{now \mapsto T_\rho(n)\}$ on a run of R_{now}.

We may now use the new clock *now* to express timing requirements of R. In particular, consider the *bounded-eventuality operator* $\Diamond_{\leq k}$, where $k \in \mathbb{N}$. The run ρ *satisfies* the bounded-eventuality formula $\Diamond_{\leq k} q$, for a past formula q, if there is a position $(n, \delta) \in P_\rho$ such that $T_\rho(n, \delta) \leq k$ and $(\rho, (n, \delta)) \models q$. The timing requirement $\Diamond_{\leq k} q$ of R is easily expressed over the elapsed-time system R_{now}:

$$R \models_{div} \Diamond_{\leq k} q \text{ iff } R_{now} \models_{div} \Diamond(q \wedge now \leq k).$$

The following proposition gives a more useful characterization of bounded eventuality, as a safety property.

Proposition 2.1 *Let R be a clock system, let q be a past formula, and let $k \in \mathbb{N}$. Then $R \models_{div} \Diamond_{\leq k} q$ iff $R_{now} \models_{div} \Box(now > k \Rightarrow \Diamond\!\!\!-(q \wedge now \leq k))$.* ■

Therefore, proving bounded eventualities reduces to proving $\Box$ formulas.

Two of the most important classes of timing requirements are *bounded response* and *bounded invariance* [HMP91], corresponding to the formulas

$$\Box(p \Rightarrow \Diamond_{\leq k} q) \text{ and } \Box(p \Rightarrow \Box_{<k} \neg q),$$

where p and q are past formulas, and $k \in \mathbb{N}$. The former states that whenever p is true at some position along a run, there is a position no more than k time units later at which q is true. The latter states that whenever p is true, then q is false at every position less than k time units in the future. Formally, the run ρ *satisfies* the bounded-response formula $\Box(p \Rightarrow \Diamond_{\leq k} q)$ iff for every position (m, δ) such that $(\rho, (m, \delta)) \models p$, there exists a position $(n, \epsilon) \geq (m, \delta)$ with $T_\rho(n, \epsilon) - T_\rho(m, \delta) \leq k$ and $(\rho, (n, \epsilon)) \models q$. The run ρ *satisfies* the bounded-invariance formula $\Box(p \Rightarrow \Box_{<k} \neg q)$ iff for every position (m, δ) such that $(\rho, (m, \delta)) \models p$, and every position $(n, \epsilon) \geq (m, \delta)$ with $T_\rho(n, \epsilon) - T_\rho(m, \delta) < k$, $(\rho, (n, \epsilon)) \models \neg q$. In order to express these timing requirements of a system within our logic, we need to add additional auxiliary variables to the system.

Let $R = (\varphi_0, \Psi, X)$ be a clock system over the set V of data variables and the set C of clocks. Let $fl_p \notin V$ be a new boolean data variable, and let $c_p \notin C$ be a new clock. Define $\varphi_p = (\varphi_0 \wedge \neg fl_p \wedge c_p = 0)$. For each guarded command $(\varphi, f) \in \Psi$, define $(\varphi, f)_p = (\varphi, f \cup \{fl_p \mapsto fl_p, c_p \mapsto c_p\})$; that is, $(\varphi, f)_p$ acts as (φ, f) on $V \cup C$, and leaves both fl_p and c_p unchanged. Let *begin* be the guarded command $(p \wedge \neg fl_p, \{x \mapsto x \mid x \in V \cup C\} \cup \{fl_p \mapsto true, c_p \mapsto 0\})$. The guarded command *begin* may be executed at most once on any run; it sets the flag fl_p to true and resets the clock c_p. Let $\Psi_p = \{begin\} \cup \{(\varphi, f)_p \mid (\varphi, f) \in \Psi\}$. Let R_p be the clock system (φ_p, Ψ_p, X) over the set $V \cup \{fl_p\}$ of data variables and the set $C \cup \{c_p\}$ of clocks. Bounded-response and bounded-invariance requirements of R can be expressed by temporal formulas over R_p.

Proposition 2.2 *Let R be a clock system, let p and q be past formulas, and let $k \in \mathbb{N}$. Then*

1. $R \models_{div} \Box(p \Rightarrow \Diamond_{\leq k} q)$ *iff* $R_p \models_{div} \Box((fl_p \wedge c_p > k) \Rightarrow \Diamond\!\!\!-(q \wedge fl_p \wedge c_p \leq k))$;
2. $R \models_{div} \Box(p \Rightarrow \Box_{<k} \neg q)$ *iff* $R_p \models_{div} \Box((fl_p \wedge c_p < k) \Rightarrow \neg q)$.

Proof. For each divergent run ρ of R, there is one run of R_p for each position (n, δ) such that $(\rho, (n, \delta)) \models p$. For at any such position, and only once during each run, the command *begin* may be executed. Denote by $\rho^{(n,\delta)}_{begin}$ the run of R_p obtained from ρ by execution of *begin* at position (n, δ).

If $(\rho,(n,\delta)) \not\models p \Rightarrow \diamond_{\leq k} q$ then $\rho^{(n,\delta)}_{begin} \not\models \Box[(fl_p \wedge c_p > k) \Rightarrow \Diamond(q \wedge fl_p \wedge c_p \leq k)]$. Conversely, if for each position $(n,\delta) \in P_\rho$, $\rho^{(n,\delta)}_{begin} \models \Box[(fl_p \wedge c_p > k) \Rightarrow \Diamond(q \wedge fl_p \wedge c_p \leq k)]$, then $\rho \models \Box(p \Rightarrow \diamond_{\leq k} q)$. The proof of the second claim is similar. ■

Therefore, proving bounded response and bounded invariance reduces to proving $\Box$ formulas.

3 System-transforming Verification

Before we present rules for proving safety and liveness properties of the divergent runs of a clock system R, we translate R into an untimed system. This translation will be used (1) to design a liveness rule, and (2) to prove the safety rule complete. Given a clock system R, we define an untimed system $\overline{R}$—the *region system* for R—by collapsing states that are indistinguishable by state predicates.

3.1 The region construction

Clock constraints have limited power. They can determine the floor and ceiling of the value of a clock (and hence whether that value is an integer). They can do the same for the difference of any two clocks, and this is all they can accomplish.

The *fraction-combining equivalence relation* $\approx$ on $\mathbb{R}$ is defined by $x \approx y$ iff $x = y$, or $\lfloor x \rfloor = \lfloor y \rfloor$ and $x, y \notin \mathbb{Z}$. The equivalence class of a real number x is denoted $\overline{x}$, and is called the *fc-value* of x. The set of equivalence classes is denoted $\mathbb{Z}_{\rm fc}$. The elements of $\mathbb{Z}_{\rm fc}$ are

$$\ldots, (-2,-1), \{-1\}, (-1,0), \{0\}, (0,1), \{1\}, (1,2), \ldots$$

The natural order on $\mathbb{Z}_{\rm fc}$ is denoted $\preceq$. The successor relation *succ* on $\mathbb{Z}_{\rm fc}$ is defined in the natural way: $succ((-2,-1)) = \{-1\}$, $succ(\{-1\}) = (-1,0)$, etc. The set of equivalence classes that contain only nonnegative elements is denoted $\mathbb{N}_{\rm fc}$. Let $\overline{d}$ be a variable of type $\mathbb{N}_{\rm fc}$. The predicate $\overline{d} \in \mathbb{N}$ is true if $\overline{d}$ is a singleton, and false if $\overline{d}$ is an interval. If $\overline{d}$ is the singleton $\{m\}$, then $\overline{d} + \{n\}$ is the singleton $\{m+n\}$. If $\overline{d}$ is the interval $(m, m+1)$, then $\overline{d} + \{n\}$ is the interval $(m+n, m+1+n)$. Last, define $-\overline{d}$ to be $\{-x \mid x \in \overline{d}\}$, and define $\lfloor \overline{d} \rfloor$ to be $\{\lfloor x \rfloor\}$, where x is any element of $\overline{d}$.

The state predicates induce an equivalence relation $\equiv$ on states, namely, $\sigma \equiv \tau$ iff for every state predicate φ, $\sigma \models \varphi$ iff $\tau \models \varphi$. This is the *region equivalence* of [AD90].[6] A *region* is an equivalence class of $\equiv$. We assume that any two states with different data components can be distinguished by data predicates; that is, all states of a region agree on the data variables. The following proposition characterizes the regions in terms of the fc-values of each clock and each difference of two clocks.

Proposition 3.1 [AD90] *For all states σ and τ, $\sigma \equiv \tau$ iff*

1. *for every data variable $x \in V$, $\sigma(x) = \tau(x)$;*
2. *for every clock $c \in C$, $\overline{\sigma(c)} = \overline{\tau(c)}$;*
3. *for every pair of clocks $c, d \in C$, $\overline{\sigma(c) - \sigma(d)} = \overline{\tau(c) - \tau(d)}$.* ■

In other words, two states belong to the same region iff they agree on the values of all data variables, agree on the floors of all clock values, and order the fractional parts of the clock values in the same way. With this is mind, we segregate the regions into the *boundary regions*, in which some clock

[6] According to our definition, there are infinitely many equivalence classes. In [AD90] there are only finitely many, because there clock constraints are limited to using constants less than some fixed bound.

has an integral value, and the *interior regions*, in which no clock has an integral value. We write $[\sigma]$ for the region of the state σ.

We now introduce the *next-region relation* $\rightsquigarrow$ on states, corresponding to a delay during which the region changes exactly once. Define $\sigma \rightsquigarrow \tau$ if

$$\tau \not\equiv \sigma \wedge (\exists t \in \mathbb{R}^+)(\tau = \sigma + t \wedge (\forall s \in \mathbb{R}^+)(s < t \Rightarrow (\sigma + s \equiv \sigma \vee \sigma + s \equiv \tau))).$$

If $\sigma \rightsquigarrow \tau$, then either $[\sigma]$ is a boundary region and $[\tau]$ is an interior region, or vice versa. Notice that $\rightsquigarrow$ is a function on the regions: if $\sigma \equiv \sigma'$, $\sigma \rightsquigarrow \tau$, and $\sigma' \rightsquigarrow \tau'$, then $\tau \equiv \tau'$. So for each region $[\sigma]$ there is a unique *successor region* $[\tau] = \{\tau \mid \sigma \rightsquigarrow \tau\}$.

Let the set of all guarded commands be denoted $\mathcal{G}$, and let the set of all guarded delays be denoted $\mathcal{D}$. An equivalence relation $\sim$ on the set of states is a (time-abstracting) *bisimulation* [ACH94] if for all states σ and τ, $\sigma \sim \tau$ implies

1. for each guarded command $(\varphi, f) \in \mathcal{G}$, if $\sigma \models \varphi$, then $\tau \models \varphi$ and $f(\sigma) \sim f(\tau)$;
2. for each guarded delay $\chi \in \mathcal{D}$ and each nonnegative real s, if σ admits χ for duration s, then there exists a nonnegative real t such that τ admits χ for duration t and $\sigma + s \sim \tau + t$.

The two states σ and τ are *bisimilar*, denoted $\sigma \equiv^b \tau$, if there exists a bisimulation $\sim$ with $\sigma \sim \tau$; that is, the bisimilarity relation $\equiv^b$ is the coarsest bisimulation. If two states are bisimilar, then they must satisfy the same state predicates. Interestingly, the converse holds as well.

Proposition 3.2 *The time-abstracting bisimilarity $\equiv^b$ coincides with the region equivalence $\equiv$.*

Proof. It is easy to verify that $\equiv$ satisfies the first condition of bisimulations. For the second condition, suppose $\sigma \equiv \tau$, and σ admits χ for duration s. Then either $\sigma \equiv \sigma + s$, in which case $\sigma + s \equiv \tau + 0$, or there is some $n \geq 1$ such that $\sigma \rightsquigarrow^n \sigma + s$. In the latter case, the result follows by induction on n from the fact that every region has a unique successor region. ■

For the remainder of this section, we fix a clock system $R = (\varphi_0, \Psi, X)$ over the set V of data variables and the set C of clocks. We construct the untimed region system $\overline{R}$ by replacing the real-valued clocks of R by discrete fc-valued variables, and by adding for each pair of clocks an fc-valued variable that records the fc-value of the clock difference.

For each clock c, the *clock surrogate* $\overline{c}$ is of type $\mathbb{N}_{\mathrm{fc}}$ (this notation should not cause confusion because the value of $\overline{c}$ will be the fc-value of c). For each pair of clocks c and d, the *clock-difference variable* Δ_{cd} is of type $\mathbb{Z}_{\mathrm{fc}}$. The variable Δ_{cd} will hold the fc-value of the difference between c and d. Let $\overline{C}$ be the set of clock surrogates $\{\overline{c} \mid c \in C\}$, and let D be the set of clock-difference variables $\{\Delta_{cd} \mid c, d \in C\}$. Given a state σ of R, the *region projection* of σ is the state $\overline{\sigma}$ over the set $V \cup \overline{C} \cup D$ of variables such that

1. for each data variable $x \in V$, $\overline{\sigma}(x) = \sigma(x)$;
2. for each clock $c \in C$, $\overline{\sigma}(\overline{c}) = \overline{\sigma(c)}$;
3. for each pair of clocks $c, d \in C$, $\overline{\sigma}(\Delta_{cd}) = \overline{\sigma(c) - \sigma(d)}$.

Notice that by Proposition 3.1, the region projection $\overline{\sigma}$ corresponds exactly to the region $[\sigma]$: $\overline{\sigma} = \overline{\tau}$ iff $\sigma \equiv \tau$.

We now define a transformation that takes each dense-time temporal formula ϕ over $V \cup C$ to a discrete-time temporal formula $\overline{\phi}$ over $V \cup \overline{C} \cup D$ (while ϕ contains clocks, $\overline{\phi}$ does not). For a data predicate φ, define $\overline{\varphi}$ to be φ. For a clock constraint, place a bar over each symbol: $\overline{c \preceq n} = \overline{c} \preceq \overline{n}$,

$\overline{c \geq n} = \overline{c} \succeq \overline{n}$, $\overline{c \leq d + n} = \overline{c} \preceq \overline{d} + \overline{n}$, and $\overline{c \geq d + n} = \overline{c} \succeq \overline{d} + \overline{n}$. Let $\overline{\mathit{first}} = \neg \ominus \mathit{true}$. Now define $\overline{\phi}$ inductively for all temporal formulas ϕ by distributing the bar over each operator: $\overline{\neg \varphi} = \neg \overline{\varphi}$, $\overline{\varphi_1 \vee \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2}$, $\overline{\varphi_1 \mathcal{U} \varphi_2} = \overline{\varphi_1} \mathcal{U} \overline{\varphi_2}$, and $\overline{\varphi_1 \mathcal{S} \varphi_2} = \overline{\varphi_1} \mathcal{S} \overline{\varphi_2}$.

Lemma 3.3 *Let σ be a state. For every state predicate φ, $\sigma \models \varphi$ iff $\overline{\sigma} \models \overline{\varphi}$.* ■

Finally, we construct an untimed system $\overline{R}$ over the set $V \cup \overline{C} \cup D$ of variables, called the *region system* for R, such that each transition of $\overline{R}$ corresponds to the execution of a guarded command of R, or to a delay of R that either stays within a region or proceeds to the successor region. We will show that the region system $\overline{R}$ models exactly the same temporal formulas, under the discrete-time interpretation, as does the original system R, with the dense-time interpretation.

The construction of $\overline{R}$ is an exercise in coding statements about the clock values into statements about their fc-values. Since we have required the initial condition of R to imply $c = 0$ for every clock c, we define the initial condition φ_r of the region system $\overline{R}$ to be $\overline{\varphi_0} \wedge \bigwedge_{c,d \in C} (\Delta_{cd} = 0)$.

Lemma 3.4 *Let σ be a state. Then $\sigma \models \varphi_0$ iff $\overline{\sigma} \models \varphi_r$.* ■

For each guarded command $(\varphi, f) \in \Psi$, we define the action predicate $(\varphi, f)_r$ by

$$\begin{aligned}(\varphi, f)_r \quad = \quad & \overline{\varphi} \wedge \textstyle\bigwedge_{x \in V} (x' = f(x)) \wedge \bigwedge_{c \in C} (\overline{c}' = \overline{f(c)}) \wedge \\ & \textstyle\bigwedge_{c,d \in C} ((\overline{c}' = \overline{c} \wedge \overline{d}' = \overline{d}) \Rightarrow \Delta'_{cd} = \Delta_{cd}) \wedge \\ & \textstyle\bigwedge_{c,d \in C} ((\overline{d}' = 0 \Rightarrow \Delta'_{cd} = \overline{c}') \wedge (\overline{c}' = 0 \Rightarrow \Delta'_{cd} = -\overline{d}')).\end{aligned}$$

The first two conjunctions give the correct values to all data variables and clock surrogates. The latter two conjunctions make sure that each clock-difference variable Δ_{cd} correctly reflects $\overline{c - d}$.

Lemma 3.5 *Let σ and τ be two states, and let (φ, f) be a guarded command. Then $(\overline{\sigma}, \overline{\tau}) \models (\varphi, f)_r$ iff (φ, f) is enabled in σ and $\tau \equiv f(\sigma)$.* ■

Our translation of a guarded delay takes into account at most one application of the next-region relation $\rightsquigarrow$. In this way we avoid the skipping of regions that are passed by delays. For zero applications of $\rightsquigarrow$, i.e., delays that do not change the region, we use the action predicate $\mathit{idle} = \bigwedge_{x \in V \cup \overline{C} \cup D} (x' = x)$. For one application of $\rightsquigarrow$, given a guarded delay χ, we define

$$\chi_r \; = \; \mathit{nextregion} \wedge \mathit{admit}(\chi),$$

where *nextregion* is an action predicate that is satisfied by a pair $(\overline{\sigma}, \overline{\tau})$ of regions iff τ is the successor region of σ, and $\mathit{admit}(\chi)$ is a data predicate that is satisfied by a region $\overline{\sigma}$ iff any (equivalently, every) state in $\overline{\sigma}$ admits χ for a duration that results in a state of the successor region.

First we define the action predicate *nextregion*. To determine the successor region of a given state, we must determine the order in which the clocks change their fc-values as time passes. For a state σ, the set $\mathit{Crit}(\sigma)$ of *critical clocks*, which are the first to change their fc-values, is defined by

$$\mathit{Crit}(\sigma) \; = \; \{c \in C \mid (\forall d \in C)(\lceil \sigma(d) \rceil - \sigma(d) \geq \lceil \sigma(c) \rceil - \sigma(c))\}.$$

If $\overline{\sigma}$ is a boundary region, then $\mathit{Crit}(\sigma)$ is the set of clocks with integer values. If $\overline{\sigma}$ is an interior region, then $\mathit{Crit}(\sigma)$ is the set of clocks whose fractional parts are greater than or equal to the fractional parts of all other clocks.

Using $\mathit{Crit}(\sigma)$, we obtain a new characterization of the next-region relation. Let σ and τ be states of R. Then $\sigma \rightsquigarrow \tau$ iff

1. $\tau = \sigma + s$ for some $s > 0$;
2. for all $c \in Crit(\sigma)$, $\overline{\tau(c)} = succ(\overline{\sigma(c)})$;
3. for all $c \in C - Crit(\sigma)$, $\overline{\tau(c)} = \overline{\sigma(c)}$.

Therefore we define

$$nextregion \;=\; \bigwedge_{c \in C} ((critical(c) \;\Rightarrow\; \overline{c}' = succ(\overline{c})) \;\wedge\; (\neg critical(c) \;\Rightarrow\; \overline{c}' = \overline{c})),$$

where the data predicate $critical(c)$ is yet to be defined. To determine if the fractional part of the clock c is greater than the fractional part of clock d, it suffices to look at the fc-value of Δ_{cd}: for all $x, y \in \mathbb{R}^+$,

1. $x - \lfloor x \rfloor > y - \lfloor y \rfloor$ iff $\overline{x - y} = (\lfloor x \rfloor - \lfloor y \rfloor, 1 + \lfloor x \rfloor - \lfloor y \rfloor)$;
2. $x - \lfloor x \rfloor = y - \lfloor y \rfloor$ iff $\overline{x - y} = \{\lfloor x \rfloor - \lfloor y \rfloor\}$;
3. $x - \lfloor x \rfloor < y - \lfloor y \rfloor$ iff $\overline{x - y} = (\lfloor x \rfloor - \lfloor y \rfloor - 1, \lfloor x \rfloor - \lfloor y \rfloor)$.

From this we obtain a simple characterization of $Crit(\sigma)$. If $\overline{\sigma}$ is a boundary region, then $Crit(\sigma) = \{c \in C \mid \sigma(c) \in \mathbb{N}\}$. If $\overline{\sigma}$ is an interior region, then

$$Crit(\sigma) \;=\; \{c \in C \mid (\forall d \in C)(\overline{\sigma(d)} + \lfloor \overline{\sigma(c) - \sigma(d)} \rfloor = \overline{\sigma(c)})\}.$$

We use this observation to define the data predicate $critical(c)$:

$$critical(c) \;=\; (boundary \;\wedge\; \overline{c} \in \mathbb{N}) \;\vee\; \left(interior \;\wedge\; \bigwedge_{d \in C} (\overline{d} + \lfloor \Delta_{cd} \rfloor = \overline{c}) \right),$$

where *boundary* is the data predicate $\bigvee_{d \in C} (\overline{d} \in \mathbb{N})$, and *interior* is $\neg boundary$. This completes the definition of *nextregion*.

The definition of $admit(\chi)$ is simple. If the current region $\overline{\sigma}$ is an interior region, then the successor region $\overline{\tau}$ is a boundary region, and so σ admits χ for a duration that terminates in $\overline{\tau}$ iff $\sigma \models \chi$. If $\overline{\sigma}$ is a boundary region, then $\overline{\tau}$ is an interior region. In this case, for a delay to terminate inside $\overline{\tau}$, a nonzero amount of time must be spent in $\overline{\tau}$. So we must also require $\tau \models \chi$. Therefore we define $admit(\chi)$ by

$$admit(\chi) \;=\; (interior \wedge \overline{\chi}) \;\vee\; (boundary \wedge \overline{\chi} \wedge \overline{\chi}'),$$

where $\overline{\chi}'$ is the formula obtained from $\overline{\chi}$ by replacing each variable x by its primed counterpart x'. This completes the definition of the action predicate χ_r.

Lemma 3.6 *Let σ be a state, let χ a guarded delay, and let t be a positive real. Then σ admits χ for duration t and $\sigma \rightsquigarrow \sigma + t$ iff $(\overline{\sigma}, \overline{\sigma + t}) \models \chi_r$. Moreover, if τ is also a state, then $(\overline{\sigma}, \overline{\tau}) \models \chi_r$ iff there exists a positive real s such that σ admits χ for duration s, $\sigma \rightsquigarrow \sigma + s$, and $\tau \equiv \sigma + s$.* ■

We now define the *region system*

$$\overline{R} \;=\; (\varphi_r, idle \;\vee \bigvee_{(\varphi, f) \in \Psi} (\varphi, f)_r \;\vee \bigvee_{\chi \in X} \chi_r).$$

For each run $\rho = (\sigma_n)$ of R, there is a run $\mathbf{r}$ of the region system such that $\mathbf{r}$ tracks ρ; $\mathbf{r}$ is called the *projection of ρ into the region system*. For each position $(n, \delta) \in P_\rho$, we have the region projection $\overline{\rho(n, \delta)}$. The regions obtained by projecting all positions of ρ into regions form an infinite sequence. This is because even though there are uncountably many positions, the region projection changes only countably many times. The resulting sequence $\mathbf{r}$ of regions is a run of the region system $\overline{R}$.

Formally, we define $\mathbf{r}$ by induction. First, let $\mathbf{r}_0 = \overline{\sigma_0}$. Having defined $\mathbf{r}$ up to $\mathbf{r}_k$ and σ_n, if $\sigma_n \triangleright_\Psi \sigma_{n+1}$, then $\mathbf{r}_{k+1} = \overline{\sigma_{n+1}}$. If $\sigma_n \triangleright_X \sigma_{n+1}$, then either $\sigma_{n+1} \equiv \sigma_n$, in which case we define $\mathbf{r}_{k+1} = \overline{\sigma_{n+1}}$; or there is a sequence $\tau_0, \tau_1, \ldots, \tau_m$ of states such that $\sigma_n = \tau_0$, $\sigma_{n+1} = \tau_m$, and for each $i < m$, $\tau_i \rightsquigarrow \tau_{i+1}$. In this case define $\mathbf{r}_{k+i} = \overline{\tau_i}$ for $i = 1, \ldots, m$.

The projection $\mathbf{r}$ of ρ into the region system is a run of $\overline{R}$ by the previous three lemmas. Conversely, given any run $\mathbf{r}$ of $\overline{R}$, there is a run $\rho = (\sigma_n)$ of R such that $\mathbf{r}$ is the projection of ρ. Let σ_0 be any state such that $\overline{\sigma_0} = \mathbf{r}_0$ (actually, there is only one such σ, because all clocks start at 0). Then for each k, if $(\mathbf{r}_k, \mathbf{r}_{k+1}) \models (\varphi, f)_r$ for some guarded command $(\varphi, f) \in \Psi$, then let $\sigma_{k+1} = f(\sigma_k)$. If $(\mathbf{r}_k, \mathbf{r}_{k+1}) \models \chi_r$ for some guarded delay $\chi \in X$, then let σ_{k+1} be any state such that $\sigma_k \rightsquigarrow \sigma_{k+1}$. Finally, if $(\mathbf{r}_k, \mathbf{r}_{k+1}) \models \mathit{idle}$, then let $\sigma_{k+1} = \sigma_k$.

Theorem 3.7 *Let ρ be a run of R, and let $\mathbf{r}$ be the projection of ρ into the region system. Then for each temporal formula ϕ, $\rho \models \phi$ iff $\mathbf{r} \models \overline{\phi}$.*

Proof. The definition of the projection $\mathbf{r}$ of ρ provides a nondecreasing onto map $g : P_\rho \to \mathbb{N}$ such that for each position $(n, \delta) \in P_\rho$, $\overline{\rho(n, \delta)} = \mathbf{r}_{g(n,\delta)}$. By induction on ϕ, $(\rho, (n, 0)) \models \phi$ iff $(\mathbf{r}, g(n, 0)) \models \phi$ for all $n \in \mathbb{N}$. Use $n = 0$ to obtain the theorem. ■

This theorem allows us to reduce the problem of proving properties of the clock system R to proving properties of the untimed system $\overline{R}$. If we wished to prove ϕ for all runs of R, we could simply prove $\overline{\phi}$ for all runs of $\overline{R}$. In the following section, we show how to prove ϕ for all *divergent* runs of R.

We remark that the region construction works because the clock model does not allow much interaction between data variables and clocks. For example, we cannot assign the value of a clock to a data variable, for then the bisimilarity $\equiv^b$ is equality on the states. We could, however, allow the testing of clocks against arbitrary integer expressions over the data variables, and the assignment of such expressions to clocks.

3.2 Proving liveness properties of the divergent runs

The region system $\overline{R_{now}}$ of the elapsed-time system R_{now} is the natural setting for proving properties of the divergent runs of R. The run ρ of R_{now} is divergent iff the projection of ρ into $\overline{R_{now}}$ satisfies the condition $\Box\Diamond(\overline{now} \succ \ominus\overline{now})$. This observation gives the following corollary to Theorem 3.7.

Corollary 3.8 *Let R be a clock system. For every temporal formula ϕ, $R \models_{div} \phi$ iff $\overline{R_{now}} \models (\Box\Diamond(\overline{now} \succ \ominus\overline{now})) \Rightarrow \phi$.* ■

Corollary 3.8 is the basis for a proof calculus for properties of the divergent runs of R. To prove a property ϕ is satisfied by all divergent runs of R, i.e., $R \models_{div} \phi$, we may use, for example, the proof calculus of [MP89] for untimed systems to show that $(\Box\Diamond(\overline{now} \succ \ominus\overline{now})) \Rightarrow \phi$ is satisfied by the region system $\overline{R_{now}}$. The soundness and completeness of the proof calculus for the untimed system $\overline{R_{now}}$, coupled with Corollary 3.8, immediately gives soundness and completeness for proving properties of the divergent runs of R.

We give an example proof via the region system in Appendix A1.

The premise $\Box\Diamond(\overline{now} \succ \ominus\overline{now})$ used in region system proofs looks like a fairness requirement. Suppose that we group together the transitions that correspond to one application of the next-region

relation $\rightsquigarrow$, and put a strong-fairness requirement on this set SF. We say that SF is enabled if any of its members is enabled. Then we require strong fairness for SF; that is, if SF is enabled infinitely often, then infinitely many of the transitions taken will be elements of SF. While every strongly fair run of $\overline{R_{now}}$ corresponds to a divergent run of R, the set SF need not be infinitely often enabled on every run of $\overline{R_{now}}$. Therefore the divergent runs may well be a proper subset of the strongly fair runs. Hence a proof calculus for strong fairness would be incomplete for proving properties of the divergent runs.

3.3 Proving safety properties of the divergent runs

For nonzeno systems, there is a simpler way of proving safety properties than the method given in the previous section. The following observation implies that, in order to show that every divergent run of the clock system R satisfies a $\Box$ formula, it suffices to use a standard invariance argument to show that every finite run prefix of the region system $\overline{R}$ satisfies the formula.

Proposition 3.9[7] *If R is a nonzeno clock system and q is a past formula, then $R \models_{div} \Box q$ iff $\overline{R} \models \Box q$.* ■

Proof. Suppose that $R \models_{div} \Box q$. If R is nonzeno, then the set Div of runs of $\overline{R_{now}}$ that satisfy $\Box\Diamond(\overline{now} \succ \ominus\overline{now})$ is a dense subset[8] of the set all runs of $\overline{R_{now}}$. Since the set Q of runs of $\overline{R_{now}}$ that satisfy $\Box q$ is closed,[9] if $Div \subset Q$, then all runs of $\overline{R_{now}}$ satisfy $\Box q$, and hence all runs of $\overline{R}$ satisfy $\Box q$. The converse follows from Theorem 3.7. ■

An example that shows the necessity of the nonzeno condition is afforded by the clock system R with the initial condition $x = 0 \wedge c = 0$, the guarded command $(x = 0, \{x \mapsto 1, c \mapsto c\})$, and the guarded delay $x = 0$, where x is a data variable and c is a clock. When $x = 0$, then R allows either an arbitrary amount of time to pass, or the execution of the guarded command, which increments x. When $x = 1$, no time is allowed to pass. Therefore $R \models_{div} \Box(x = 0)$, but not all runs of R satisfy $\Box(x = 0)$.

Because of Proposition 3.9, if R is a nonzeno clock system, we may use any standard method for proving discrete-time safety properties to prove dense-time safety properties. For example, we may use the sound and complete rule $SAFE$ from [MP89] to prove $\overline{R} \models \Box q$. The rule $SAFE$ states that for past formulas p and q,

$$\frac{\begin{array}{l}\models \Box((\varphi_0 \wedge \mathit{first}) \Rightarrow p) \\ \models \Box(p \Rightarrow q) \\ \models \{p\}\ \left(\mathit{idle} \vee \bigvee_{(\varphi,f)\in\Psi}(\varphi, f)_r \vee \bigvee_{\chi\in X} \chi_r\right)\ \{p\}\end{array}}{\overline{R} \models \Box q}$$

The third premise is a Hoare triple that ensures whenever p is true at some point along a run, then p remains true after any single execution step. The rule is sound and complete for proving statements of the form $\overline{R} \models \Box q$. Hence by Proposition 3.9, it is sound and complete for proving statements of the form $R \models_{div} \Box q$, provided R is nonzeno.

We only indicate briefly how one might go about showing that a clock system is nonzeno. The nonzeno condition can be expressed as a branching-time formula on the corresponding region system. The clock system R is nonzeno iff

$$\overline{R_{now}} \models \forall\Box((\overline{now} \in \mathbb{N} \Rightarrow \exists\Diamond(\overline{now} \notin \mathbb{N})) \wedge (\overline{now} \notin \mathbb{N} \Rightarrow \exists\Diamond(\overline{now} \in \mathbb{N}))).$$

[7] The corresponding observation for TLA is made in [AL92].

[8] In the Cantor topology on infinite sequences.

[9] See previous footnote.

This CTL formula asserts that from each boundary region, an interior region is reachable, and vice versa. Therefore, to prove a clock system nonzeno, we may use a proof calculus for CTL on the corresponding region system.

4 Property-transforming Verification

It has been observed that many real-time properties of interest are safety properties, and can be proved by invariance arguments [AL92, Hen92, LV92]. For example, in Section 2, we formulated bounded eventuality using a $\Box$ formula. By contrast, unbounded eventuality—a $\Diamond$ formula—is a true liveness property. However, for many real-time systems, if an event is guaranteed to occur, it will always occur within a fixed amount of time. Likewise, this may become true if there is a known or unknown lower bound on the duration of delays. In both cases, unbounded eventualities can be translated into bounded eventualities.

We study (1) various classes of real-time systems and (2) various notions of divergence as to how well they accommodate a translation of $\Diamond$ formulas into $\Box$ formulas. First, we show that relative to any safe[10] set of divergent runs, a translation is always available. Second, we introduce the class of $O(1)$-bounded real-time systems, which allow the translation for the standard notion of divergence. Third, we present a wider class of real-time systems that allow the translation for an intermediate notion of divergence called *unknown-ϵ divergence* (there is an unknown lower bound ϵ on the duration of delays).

4.1 Safe notions of divergence

Let R be a clock system, and let p and q be past formulas. Let P be the set of runs of R that satisfy $\Box p$. Then P is a safe set of runs. For example, for synchronous circuits, a useful choice of P is the set of divergent runs for which 1 time unit expires between any two transitions. We write $P \models \phi$ (resp. $P \models_{div} \phi$) if every run (resp. every divergent run) in P satisfies the temporal formula ϕ. A simple argument using König's lemma shows that if $P \models \Diamond q$, then in fact there is a $k \in \mathbb{N}$ such that $P \models \Diamond_{\leq k} q$. So safe sets of runs allow the translation of $\Box$ formulas into $\Diamond$ formulas.

Unfortunately, this translation does not per se enable the proof of unbounded eventualities over safe sets of divergent runs by safety reasoning. For to prove $P \models_{div} \Diamond_{\leq k} q$, we must prove $R \models_{div} \Box p \Rightarrow \Diamond_{\leq k} q$, and the latter is not a safety property. So we need to be more subtle. In the sequel we concentrate on other interesting (and not necessarily safe) subsets of the divergent runs for which the translation of $\Box$ properties into $\Diamond$ properties, when possible, bears fruit.

4.2 A hierarchy of real-time systems

We classify real-time systems according to how "safe" unbounded eventuality requirements are. $O(1)$-bounded systems establish all eventualities within constant time. Now suppose that no more than n delays, for some unknown constant n, may take place in one time unit. Then $O(n)$-bounded systems establish all eventualities within time $O(n)$, $O(n^2)$-bounded systems establish all eventualities within time $O(n^2)$, etc. We obtain an infinite strict hierarchy of real-time systems in this manner.

$O(1)$-bounded real-time systems. The clock system R is *$O(1)$-bounded* if for every past formula p, if $R \models_{div} \Diamond p$, then there is a constant k such that $R \models_{div} \Diamond_{\leq k} p$. Thus, by Proposition 2.1,

[10] Closed in the Cantor topology.

Program *UpDown*:
$x, y : \mathbb{N}$;
c : Clock;
initially $x = 1 \wedge c = 0$;
ℓ_1: **while** $c < 1$ **do**
ℓ_2: $x := x + 1$;
ℓ_3: $c < 1 \rightarrow$ **wait**
od;
ℓ_4: **while** $x > 0$ **do**
ℓ_5: $x := x - 1$;
ℓ_6: **wait** $[1, 1]$
od.

Figure 2: A $\Theta(n)$-bounded clock program

to prove the unbounded-eventuality formula $\Diamond p$ of an $O(1)$-bounded real-time system, we run the proof of a safety property of the elapsed-time system R_{now} with k as a symbolic constant. Such a proof is given in Appendix A2.

Example 4.1 Every timed safety automaton is $O(1)$-bounded [HNSY92]. This is due to the finite data state space of timed safety automata. ■

$O(f)$-bounded real-time systems. Not all clock systems are $O(1)$-bounded. Consider the clock program *UpDown* from Figure 2. A *clock program* is a while program that declares certain variables to be of the type "Clock" and contains guarded wait instructions. Every clock program defines a clock system (for details, we refer the reader to [AH93]). In particular, the guarded wait instruction ℓ: $\chi \rightarrow$ **wait** specifies the guarded delay $at\_\ell \wedge \chi$; that is, at the control location ℓ, a delay is permitted that must end as soon as the guard χ becomes false, but may end before this occurs. We use the shorthand **wait** $[l, u]$ for the program fragment $c := 0$; **while** $c < l$ **do** $c < u \rightarrow$ **wait od**, where c is a new clock; that is, a delay of any duration between l and u is permitted. Before termination, all delays of a clock program are explicitly specified by guarded wait instructions (after termination, any delays are permitted).

UpDown $\models_{div} \Diamond(x = 0)$, because every divergent run passes through both while loops. The first loop may be executed arbitrarily many times (for an accumulated duration of 1 time unit). Therefore arbitrarily many iterations of the second loop, each of which takes 1 time unit, may be required to decrease x to 0. Notice that if there is a lower bound ϵ on the length of a delay, then a state with $x = 0$ is reached within $2 + \lceil 1/\epsilon \rceil$ time units. This brings us to the following definitions.

Let R be a clock system and let ϵ be a positive real. The run ρ of R is *ϵ-divergent* if ρ is divergent and all delays in ρ have a duration of at least ϵ. We write $R \models_{div_\epsilon} \phi$ if every ϵ-divergent run of R satisfies ϕ. The run ρ is *unknown ϵ-divergent* if ρ is ϵ-divergent for some $\epsilon > 0$; that is, the durations of the delays in ρ do not decrease without bound. One may argue that, in practice, the set of unknown-ϵ divergent runs is just as interesting as the set of divergent runs. This is because we may well know that a real-time system admits only delays of a certain unknown duration ϵ, and the existence of such an ϵ is preserved under the composition of systems.[11] We write $R \models_{udiv} \phi$ if every unknown ϵ-divergent run of R satisfies ϕ.

[11] A related model, which assumes an unknown *upper* bound on the duration of delays, is studied in [AAT94, AH94].

Program $UpDown_f$:
  $x, y : \mathbb{N}$;
  c : Clock;
  initially $x = 1 \wedge y = 0 \wedge c = 0$;
  while $c < 1$ **do**
    $c < 1 \rightarrow$ **wait**
    $y := y + 1$
  od
  $x := f(y)$;
  while $x > 0$ **do**
    $x := x - 1$;
    wait $[1, 1]$
  od.

Figure 3: A $\Theta(f)$-bounded clock program

Let $f : \mathbb{N} \to \mathbb{N}$. The clock system R is $O(f)$-*bounded* if for every past formula p, if $R \models_{udiv} \Diamond p$, then there is a function $g \in O(f)$ such that for every $\epsilon > 0$, $R \models_{div_\epsilon} \Diamond_{\leq g(\lceil 1/\epsilon \rceil)} p$. If R is $O(f)$-bounded, then every $\Diamond$ formula is a bounded eventuality, and hence a safety property, of the ϵ-divergent runs. The bound on the eventuality is given as a function $g(\lceil 1/\epsilon \rceil)$, where $g \in O(f)$. The clock system R is $\Theta(f)$-*bounded* if it is $O(f)$-bounded, but not $O(g)$-bounded for any $g \in O(f)$ with $f \notin O(g)$. The system R is *unbounded* if it is not $O(f)$-bounded for any f.

Example 4.2 The program *UpDown* of Figure 2 is $\Theta(n)$-bounded. If all delays have duration at least ϵ, then the program terminates within $2 + \lceil 1/\epsilon \rceil$ time units, and so *UpDown* is $O(n)$-bounded. If all delays have duration exactly ϵ, then the program terminates in exactly $2 + \lceil 1/\epsilon \rceil$ time units, and so *UpDown* is $\Theta(n)$-bounded.

Proposition 4.3 *For every computable function f there is a $\Theta(f)$-bounded clock system.*

Proof. Consider the clock program $UpDown_f$ of Figure 3. Suppose that all delays have duration at least ϵ. Then there are at most $\lceil 1/\epsilon \rceil$ iterations of the first loop. Therefore at most $f(\lceil 1/\epsilon \rceil)$ time is spent in the second loop. The total time spent in the first loop is 1 unit, and so $UpDown_f$ terminates within $1 + f(\lceil 1/\epsilon \rceil)$ time units. Therefore $UpDown_f$ is $O(f)$-bounded. If each delay takes exactly ϵ time units, then $UpDown_f$ terminates in exactly $1 + f(\lceil 1/\epsilon \rceil)$ time units. Therefore $UpDown_f$ is $\Theta(f)$-bounded. ■

Suppose that we know the clock system R is $O(f)$-bounded, for a given function f. Then, to prove $R \models_{udiv} \Diamond p$, we use Proposition 2.1 and run a proof of the safety property $\Diamond_{\leq kf(\lceil 1/\epsilon \rceil)} p$ in the elapsed-time system R_{now}, with a symbolic constant k. We give an example of such a proof in Appendix A3.

Unbounded real-time systems. Not every clock system is $O(f)$-bounded for some f. This is because fairness can be implemented by divergence, resulting in an unbounded system. Consider the parallel program *FairUpDown* from Figure 4. This program begins by spawning two processes, one which continually increments x until *flag* becomes false, and one which sets *flag* to false. No time is allowed to pass until the two parallel processes exit. Consequently $FairUpDown \models_{div} \Diamond(x = 0)$. However, x may have attained any value before the second loop is entered, and so the time of termination is not bounded by any function of ϵ.

```
Program FairUpDown:
    x : ℕ;
    c : Clock;
    initially flag = true ∧ x = 1;
    cobegin
        while flag do        ‖    flag := false
            x := x + 1
        do
    coend;
    while x > 0 do
        x := x − 1;
        wait [1,1]
    od.
```

Figure 4: An unbounded clock program

A more thorough investigation of the hierarchy of $O(f)$-bounded systems is desirable. In particular, finding the bound function f for a given clock system, if such an f exists, is an interesting open problem. This is because the bound function must be known to apply the symbolic-constant proof technique for transforming eventualities into invariances over the ϵ-divergent runs.

Acknowledgements. We thank Limor Fix for helpful discussions.

Appendix

A.1 A region-system eventuality proof

Consider the program *Wait5* from the introduction, which loops waiting until five units of time pass. We prove that $\mathit{Wait5} \models_{div} \Diamond(x = 0)$.

The program *Wait5* translates into a clock system $R = (\varphi_0, \Psi, X)$. If we let ℓ be the program counter, then the initial condition φ_0 is $\ell = \ell_1 \wedge x = 1 \wedge c = 0$. The transition condition Ψ consists of the four guarded commands

$$
\begin{array}{l}
(\ell = \ell_1 \wedge c < 5, \{\ell \mapsto \ell_2,\ x \mapsto x,\ c \mapsto c\}),\\
(\ell = \ell_1 \wedge c \geq 5, \{\ell \mapsto \ell_3,\ x \mapsto x,\ c \mapsto c\}),\\
(\ell = \ell_2, \{\ell \mapsto \ell_1,\ x \mapsto x,\ c \mapsto c\}),\\
(\ell = \ell_3, \{\ell \mapsto \ell_4,\ x \mapsto 0,\ c \mapsto c\}).
\end{array}
$$

Here we model with location ℓ_4 a state from which only delays are allowed. The environment condition X consists of the two guarded delays $\ell = \ell_2 \wedge c < 5$ and $\ell = \ell_4$.

We now prove that $R \models_{div} \Diamond(x = 0)$ using the region system $\overline{R}$. Since the clock c measures elapsed time there is no need to form the elapsed-time system R_{now}. The initial condition of the region system is $\overline{\varphi_0}$. Next we construct $\psi = \bigvee_{(\varphi,f)\in\Psi}(\varphi, f)_r$ as

$$
\begin{array}{l}
(\ell = \ell_1 \wedge \overline{c} \prec \overline{5} \wedge \ell' = \ell_2 \wedge x' = x \wedge \overline{c}' = \overline{c}) \vee\\
(\ell = \ell_2 \wedge \ell' = \ell_1 \wedge x' = x \wedge \overline{c}' = \overline{c}) \vee\\
(\ell = \ell_1 \wedge \overline{c} \succeq \overline{5} \wedge \ell' = \ell_3 \wedge x' = x \wedge \overline{c}' = \overline{c}) \vee\\
(\ell = \ell_3 \wedge \ell' = \ell_4 \wedge x' = 0 \wedge \overline{c}' = \overline{c}).
\end{array}
$$

For the guarded delay $\chi = (\ell = \ell_2 \wedge c < 1)$ we obtain

$$\chi_r \;=\; (\ell = \ell_2 \wedge \overline{c} \prec \overline{5} \wedge \overline{c}' = succ(\overline{c}) \wedge x' = x \wedge \ell' = \ell).$$

For the guarded delay $\xi = (\ell = \ell_4)$ we obtain

$$\xi_r \;=\; (\ell = \ell_4 \wedge \overline{c}' = succ(\overline{c}) \wedge x' = x \wedge \ell' = \ell).$$

The region system $\overline{R}$ is then $(\overline{\varphi_0}, \psi_\triangleright)$ for the transition condition $\psi_\triangleright = (\psi \vee \chi_r \vee \xi_r \vee idle)$.

To prove that $\Diamond(x = 0)$ holds in R, we must prove that $\overline{R}$ satisfies the temporal formula

$$(\Box\Diamond(\overline{c} \succ \ominus\overline{c})) \Rightarrow \Diamond(x = 0).$$

To do so, we must find a convenient well-founded ordering W. Here we need 10 changes in the value of c before the desired event occurs, and so we choose $W = \{0, 1, 2, \ldots, 10\}$ with the natural order. We use the proof rule *B-REAC* from [MP89]. It states that given a function ω mapping nonempty finite sequences of states into W, past formulas p, q, and r, and any temporal formula ϕ, if

1. $\overline{R} \models \Box(p \Rightarrow (q \vee \phi))$
2. $\models (\phi \wedge \omega = i \wedge \psi_\triangleright) \Rightarrow (q \vee (\phi \wedge \omega \leq i))'$
3. $\overline{R} \models \Box((\phi \wedge \omega = i \wedge r) \Rightarrow \Diamond(q \vee (\phi \wedge \omega < i)))$

then we may conclude that $\overline{R} \models \Box((p \wedge \Box\Diamond r) \Rightarrow \Diamond q)$. Notice that ω appears as part of a temporal formula, and so ω must be expressible in the logic. We will now define the formulas $\omega = i$ for $i \in \{0, 1, 2, \ldots, 10\}$. Then $\omega < i$ is $\bigvee_{j<i}(\omega = j)$. For $i > 0$, define $\omega = i$ to be $\overline{c} = succ^{10-i}(\overline{0})$, and define $\omega = 0$ to be $\overline{c} \succeq \overline{5}$.

Inspecting the format of the proof rule, we put $\phi = (\ell \neq \ell_4)$, $p = \varphi_0$, $q = (x = 0)$, and $r = (\overline{c} \prec \bigcirc\overline{c})$. The first premise is

$$\Box((\ell = \ell_1 \wedge x = 1 \wedge \overline{c} = \overline{0}) \Rightarrow (x = 0 \vee \ell \neq \ell_4)),$$

which is valid. The action predicates

$$(\phi \wedge \omega = i \wedge \psi_\triangleright) \Rightarrow (q \vee (\phi \wedge \omega \leq i))'$$

are valid for each $i \in 0, 1, 2, \ldots, 10$. Finally, the third premise is

$$\Box((\ell \neq \ell_4 \wedge \overline{c} = \overline{i} \wedge \overline{c} \prec \bigcirc\overline{c}) \Rightarrow \Diamond(x = 0 \vee (\ell \neq \ell_4 \wedge \overline{c} \succ \overline{i}))).$$

This is true in $\overline{R}$ for each i, by the invariance rule *SAFE*.

A.2 A symbolic-constant eventuality proof for an $O(1)$-bounded real-time system

The program *Wait5* is $O(1)$-bounded and nonzeno. We now prove that *Wait5* $\models_{div} \Diamond_{\leq k}(x = 0)$, using k as a symbolic constant. The proof will allow us to decide which values of k can be chosen to satisfy the claim. Define I to be

$$\begin{array}{l}(\ell = \ell_1 \Rightarrow (x = 1 \wedge c \leq 5)) \wedge (\ell = \ell_2 \Rightarrow (x = 1 \wedge c \leq 5)) \wedge \\ (\ell = \ell_3 \Rightarrow (x = 1 \wedge c = 5)) \wedge (\ell = \ell_4 \Rightarrow (x = 0 \wedge c \geq 5)),\end{array}$$

where ℓ_4 corresponds to the end of the program, when time passes forever. The rule *SAFE* proves *Wait5* $\models_{div} \Box I$. Now we choose k so that $I \Rightarrow (c \geq k \Rightarrow x = 0)$ is valid. To do so, we falsify $c \geq k$ except where I implies $x = 0$. Each of the locations at which I does not imply $x = 0$ provides an upper bound of 5 for c. So choosing $k > 5$, we obtain the validity of $I \Rightarrow (c \geq k \Rightarrow x = 0)$. Since *Wait5* $\models_{div} \Box I$, we have *Wait5* $\models_{div} \Diamond_{\leq k}(x = 0)$ when $k > 5$.

A.3 A symbolic-constant eventuality proof for an $O(n)$-bounded real-time system

We prove $UpDown \models_{div_\epsilon} \Diamond_{\leq k/\epsilon}(x = 0)$, where k is a symbolic constant. The program location ℓ_7 corresponds to the end of the program. Define J to be

$$\begin{array}{l}(\ell = \ell_1 \Rightarrow (c \geq \epsilon(x-1) \wedge c \leq 1 \wedge x \geq 0)) \wedge \\ (\ell = \ell_2 \Rightarrow (c \geq \epsilon(x-1) \wedge c < 1 \wedge x \geq 0)) \wedge \\ (\ell = \ell_3 \Rightarrow (c \geq \epsilon(x-2) \wedge c \leq 1 \wedge x \geq 0)) \wedge \\ (\ell = \ell_4 \Rightarrow (1/\epsilon \geq x + c - 2 \wedge x \geq 0)) \wedge \\ (\ell = \ell_5 \Rightarrow (1/\epsilon \geq x + c - 2 \wedge x > 0)) \wedge \\ (\ell = \ell_6 \Rightarrow (1/\epsilon \geq x + c - 1 \wedge x > 0)) \wedge \\ (\ell = \ell_7 \Rightarrow \Diamond\!\!\!-(1/\epsilon \geq c - 2 \wedge x = 0)).\end{array}$$

$UpDown \models_{div_\epsilon} \Box J$, as is easily seen by the rule *SAFE*, modified so as to accomodate no delays of length less than ϵ (let p be J itself).

We now proceed as in the previous example, picking k large enough so that $J \Rightarrow (c \geq k/\epsilon \Rightarrow x = 0)$ is valid. We falsify $c \geq k/\epsilon$ except where J implies $x = 0$. For ℓ_1, ℓ_2, and ℓ_3, we have $1 \geq c$, and so choosing $k > \epsilon$ suffices. For ℓ_4, ℓ_5, and ℓ_6, we have $1/\epsilon \geq x + c - 2$ and $x \geq 0$, and so $1/\epsilon + 2 \geq c$. Consequently, choosing $k > 1 + 2\epsilon$ suffices. The latter number is larger, and so if $k > 1 + 2\epsilon$, then $J \Rightarrow (c \geq k/\epsilon \Rightarrow x = 0)$ is valid. Since $UpDown \models_{div_\epsilon} \Box J$, we have $UpDown \models_{div_\epsilon} (c \geq k/\epsilon \Rightarrow \Diamond\!\!\!-(x = 0))$.

References

[AAT94] R. Alur, H. Attiya, and G. Taubenfeld. Time-adaptive algorithms for synchronization. In *Proceedings of the 26th Annual Symposium on Theory of Computing*. ACM Press, 1994.

[ACH94] R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In B. Jonsson and J. Parrow, editors, *CONCUR 94: Concurrency Theory*, Lecture Notes in Computer Science 836, pages 162–177. Springer-Verlag, 1994.

[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.

[AH92] 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.

[AH93] R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. In T. Rus, editor, *Proceedings of the First AMAST Workshop on Real-time Systems*, 1993. To appear.

[AH94] R. Alur and T.A. Henzinger. Finitary fairness. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science*, pages 52–61. IEEE Computer Society Press, 1994.

[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.

[GSSAL93] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. Technical Report 587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1993.

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

[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.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science*, pages 394–406. IEEE Computer Society Press, 1992.

[LV92] N.A. Lynch and F. Vaandrager. Forward and backward simulations for timing-based systems. 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 397–446. Springer-Verlag, 1992.

[MP89] Z. Manna and A. Pnueli. Completing the temporal picture. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *ICALP 89: Automata, Languages, and Programming*, Lecture Notes in Computer Science 372, pages 534–558. Springer-Verlag, 1989.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[MP93a] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30(2):609–678, 1993.

[MP93b] Z. Manna and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 4–35. Springer-Verlag, 1993.

[Schar] F.B. Schneider. *On Concurrent Programming*. To appear.