Nordic Journal of Computing 10(2003), 1–35.

TIMED CSP = CLOSED TIMED ε-AUTOMATA

JOËL OUAKNINE
Computer Science Department, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh PA 15213
U.S.A.
`joelo@andrew.cmu.edu`

JAMES WORRELL
Department of Mathematics, Tulane University
6823 St. Charles Ave., New Orleans LA 70118
U.S.A.
`jbw@math.tulane.edu`

Abstract. We propose some mild modifications to the syntax and semantics of Timed CSP which significantly increase expressiveness. As a result, we are able to capture some of the most widely used specifications on timed systems as refinements (reverse inclusion of sets of behaviours) which may then be verified using the model checker FDR. We characterize the expressive power of the finite-state fragment of this augmented version of Timed CSP as that of closed timed ε-automata—timed automata with ε-transitions (silent transitions) and closed invariant and enabling clock constraints.

ACM CCS Categories and Subject Descriptors: F.1.1 [Computation by Abstract Devices]: Models of Computation

Key words: Timed CSP, timed automata, denotational and operational semantics, verification, digitization

1. Introduction

The formal analysis of real-time systems usually involves both an implementation formalism and a specification formalism, together with a mechanism for deciding whether a particular implementation meets a given specification. For example, one may choose the framework of timed automata [3, 4] as implementation formalism, and some (quantitative) temporal logic to express specifications [6]. Verification can then be carried out using model checking [2].

In the case of Reed and Roscoe's dense-time process algebra Timed CSP [29, 28, 30, 35], specifications usually consist of allowable sets of behaviours. Such specifications can be represented directly [32, 12], or using a formal specification language such as Z [33]. Verifying that processes meet their specifications then usually proceeds through some kind of proof system [32, 13, 12, 33]. Naturally, one can also use temporal logics, translate processes into timed automata, and revert to the technique described in the previous paragraph; this has been accomplished (subject to various restrictions) in [18].

Received October 15, 2002; revised March 15, 2003; accepted March 21, 2003.

In the case of dense-time modelling paradigms, there have been very few attempts to express specifications using the implementation framework, and satisfaction as refinement (reverse inclusion of sets of behaviours).[1] The only instance we are aware of is the use of Alur *et al.*'s event-clock automata to express specifications of (arbitrary) timed automata [5]. More recently, we have shown in the context of Timed CSP that discrete refinement techniques could be used to verify specifications on certain dense-time systems using the model checker FDR[2] [23, 24].

There are a number of benefits and drawbacks in expressing specifications as refinements, but a full discussion of the matter is beyond the scope of this paper. For our purposes, one of the main advantages of a refinement-based approach for Timed CSP is that verification can be carried out on the model checker FDR, as discussed in the references cited above. The main disadvantage of a refinement-based approach for Timed CSP is that expressiveness for the purposes of specification was, until now, spectacularly poor.

This problem is not an intrinsic feature of a refinement approach, but is rather an artefact of Reed and Roscoe's version of Timed CSP. For instance, to express the specification 'the process P cannot perform the event a', one must show that P refines $RUN_{\Sigma-\{a\}}$, where $RUN_{\Sigma-\{a\}}$ is a process which can communicate any sequence of events other than a's. The semantic assumption of well-timedness made by Reed and Roscoe (required for processes to be well-defined) however bans such arbitrarily speedy (or Zeno) processes.

A natural solution is therefore to attempt to ease some of the restrictions imposed on the language in order to obtain a more expressive version of Timed CSP, one in which such processes, able to communicate arbitrarily many events at any given point in time, are allowed. Another desirable feature is the addition of signals to the language, to be able to express specifications such as 'the process must perform an a within two time units'. ('Soft' signals were incorporated into Timed CSP in [12]; the 'hard' approach we propose, which potentially introduces timestops, differs in important respects.) Naturally, it is highly desirable that the path we follow retain sufficiently robust ties to CSP to preserve the use of such techniques as FDR model checking.

We therefore end up with a model in which processes may exhibit any of the following properties:

- *Livelock*: the process is stuck in an infinite sequence of τ-transitions. We treat this situation as catastrophic, and do not attempt to extract any meaningful information from such processes. In practice, we have developed techniques which can guarantee that a process is livelock-free [25].

- *Timestop*: the process cannot communicate any event nor let time pass—this results from inconsistent timing requirements.

[1] Lamport has proposed an interesting dual approach using TLA$^+$, in which implementations are translated into logical formulas of the same type as those used to express specifications; satisfaction then reduces to logical implication [36].

[2] FDR is a commercial product of Formal Methods (Europe) Ltd.

- *Strong Zenoness*: the process is forced to communicate an infinite number of visible events (signals) in a finite amount of time (usually a single instant), and can therefore not let time advance past a certain point.
- *(Weak) Zenoness*: the process is capable of communicating an infinite number of events in a finite amount of time (but is also capable of letting time elapse).
- *Finite variability*: the process may only communicate finitely many events in a finite amount of time; previous treatments of Timed CSP required all processes to have this property.

This augmented version of Timed CSP turns out to be powerful enough to capture the most widely used specifications on timed systems as refinements between Timed CSP processes. Moreover, such refinements can be verified on the model checker FDR by means of digitization techniques [15]. In fact, we even show that a number of branching-time liveness properties such as timestop-freedom and constant availability can be verified through digitization (and FDR), in contrast to the situation with timed automata.

We characterize the expressive power of the finite-state fragment of this new version of Timed CSP as that of closed timed ε-automata. Closed timed ε-automata are the timed safety automata of [16], augmented with ε-transitions (silent transitions) [3, 8], and with exclusively closed invariant and enabling clock constraints (of the form $x \leqslant 3$ rather than $x < 3$, for example). In this way, we relate very precisely two major paradigms for representing, and reasoning about, timed systems. One consequence of this correspondence is that most analysis and verification techniques applicable to one model can be carried over to the other model.

Timed CSP appears to be the most general modelling formalism systematically yielding processes closed under digitization (and thus amenable to digitization techniques), making it a prime candidate for the practical formal analysis of timed systems.

1.1 Structure of the paper

The next section introduces Timed CSP syntax. This includes both standard Timed CSP operators (without unbounded nondeterminism), as well as two new operators, one introducing urgency for visible events (to model signals), and the other denoting a process which waits a nondeterministic amount of time, then terminates (needed to express certain key specifications).

Section 3 then presents a denotational semantics for our chosen Timed CSP syntax. Somewhat surprisingly, it is necessary to depart significantly from the standard timed failures model in order to obtain a satisfactory compositional semantics. We motivate our construction in great detail, and record the relevant facts which are used later on.

Next, Section 4 gives a congruent operational semantics, offering a second way to reason about processes. Although we have found it convenient to state and prove many results in terms of the operational semantics, both semantics are used throughout the paper.

A fundamental property enjoyed by Timed CSP processes is that of *closure under digitization*. This remarkable fact enables us, in many instances, to invoke powerful digitization techniques, which reduce dense-time refinement questions to discrete time. Section 5 reviews the subject of digitization, especially as it applies to Timed CSP processes.

In Section 6, we tackle the problem of expressing specifications (on Timed CSP processes) as refinements (between Timed CSP processes). We show that the most widely used specifications can be expressed as refinements, and moreover (thanks to digitization) that verification can (under certain conditions) be performed on the model checker FDR.

Having completed this presentation and study of our augmented version of Timed CSP, we introduce closed timed ε-automata in Section 7. Mirroring our treatment of Timed CSP, we give new operational and denotational semantics to these timed automata, illustrating and justifying our definitions with the help of several examples. We also recall certain standard constructions and results, such as those pertaining to the (untimed) region automaton associated with a given timed automaton.

In Section 8, we describe in considerable detail how a given closed timed ε-automaton can be represented as a (finite-state) Timed CSP process. We then derive a number of results concerning the relationship between the verification of specifications on timed automata and Timed CSP processes.

Section 9 studies the flip side of the coin, namely how it is possible to represent any finite-state Timed CSP process as a closed timed ε-automaton. We also present some interesting conjectures concerning the expressiveness of certain weakened variants of Timed CSP and timed automata.

Lastly, Section 10 concludes by summarizing the main results of our work and presenting some avenues for future research.

2. Timed CSP syntax and conventions

Let Σ be a finite set of *events*, with $\checkmark \notin \Sigma$. We write $\Sigma^{\checkmark}$ to denote $\Sigma \cup \{\checkmark\}$. In the notation below, we have $a \in \Sigma$ and $A \subseteq \Sigma$. The parameter t ranges over the non-negative reals $\mathbb{R}^+$. R denotes a (renaming) relation on Σ. Its lifting to $\Sigma^{\checkmark}$ is understood to relate $\checkmark$ to itself. The variable X is drawn from a fixed infinite set of process variables *VAR*.

Timed CSP terms are constructed according to the following grammar:

$$\begin{aligned} P ::= &\ \mathit{STOP} \mid \mathit{TIMESTOP} \mid a \longrightarrow P \mid a \xrightarrow{!} P \mid \mathit{SKIP} \mid \mathit{RANDOM} \mid \\ &\ P_1 \overset{t}{\rhd} P_2 \mid P_1 \overset{t}{\lightning} P_2 \mid P_1 \Box P_2 \mid P_1 \sqcap P_2 \mid P_1 \underset{A}{\|} P_2 \mid P_1 \,;\, P_2 \mid \\ &\ P \setminus A \mid P[R] \mid X \mid \mu X \centerdot P \mid \mathit{DIV} \ . \end{aligned}$$

STOP is the deadlocked process which is only capable of letting time pass. *TIMESTOP* is similar to *STOP* except that time itself is also blocked; it represents inconsistent timing requirements. The prefixed process $a \longrightarrow P$ initially offers at any time to engage in the event a, and subsequently behaves like P.

The signalling prefixed process $a \xrightarrow{!} P$ communicates a immediately and subsequently behaves like P. *SKIP* represents successful termination, and is willing to communicate $\checkmark$ at any time. *RANDOM* nondeterministically waits for a real-valued amount of time, and then becomes *SKIP*. $P_1 \overset{t}{\rhd} P_2$ is the timeout process that initially offers to become P_1 for t time units, after which it silently becomes P_2 if P_1 has failed to communicate any visible event. $P_1 \overset{t}{\lightning} P_2$ is the interrupt process that behaves like P_1 for the first t time units, and then silently starts behaving like P_2 (assuming P_1 has not successfully terminated in the meantime). $P_1 \sqcap P_2$ denotes the nondeterministic choice between P_1 and P_2, whereas $P_1 \Box P_2$ represents the deterministic alternative. $P_1 \Box P_2$ is willing to behave either like P_1 or like P_2, the decision being taken on the first visible event. The parallel composition $P_1 \underset{A}{\parallel} P_2$ requires P_1 and P_2 to synchronize on all events in A, and to behave independently of each other with respect to all other events. $P_1 \fatsemi P_2$ is the sequential composition of P_1 and P_2: it denotes a process which behaves like P_1 until P_1 chooses to terminate (silently), at which point the process seamlessly starts to behave like P_2. $P \setminus A$ is a process which behaves like P but with all communications in the set A hidden; the assumption of *τ-urgency* dictates that no time can elapse while hidden events are on offer—in other words, hidden events happen as soon as they become available. The renamed process $P[R]$ derives its behaviours from those of P in that, whenever P can perform an event a, $P[R]$ can engage in any event b such that $a \; R \; b$. The recursion $\mu X \centerdot P$ represents the most nondeterministic solution to the equation $X = P$ (as it turns out, this solution is unique if it is livelock-free). Lastly, the process *DIV* represents livelock, i.e., a process caught in an infinite loop of τ-transitions.

We occasionally use the following derived constructs: We abbreviate $a \longrightarrow STOP$ as simply a. The term $WAIT\ t$ denotes $STOP \overset{t}{\rhd} SKIP$. In the case of hiding a single event a, we write $P \setminus a$ rather than $P \setminus \{a\}$. If $S = \{P_1, P_2, \ldots, P_k\}$ is a finite nonempty set of processes, $\bigsqcap S$ represents $P_1 \sqcap P_2 \sqcap \ldots \sqcap P_k$, and similarly for $\Box S$. The *interleaving* operator $|||$ denotes parallel composition over an empty interface. Lastly, we usually express recursions by means of the equational notation $X = P$, rather than the functional $\mu X \centerdot P$.

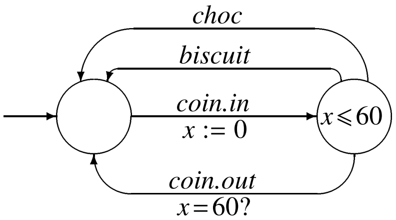
R 1. The main syntactic changes, with respect to standard presentations of Timed CSP [29, 28, 30, 35], are the elimination of *unbounded nondeterminism* (to allow implementation on model checking tools), and the twin additions of signals ($a \xrightarrow{!} P$) and the *RANDOM* process. While the latter incorporates a restricted form of unbounded nondeterminism, it is harmless insofar as digitization techniques are concerned since its integral-time semantics is finite-state (equivalent to $RANDOM = SKIP \overset{0}{\rhd} WAIT\ 1 \fatsemi RANDOM$); moreover, *RANDOM* provides us with just enough expressive power to capture certain key specification processes, as we shall see later on.

We write *TCSP* to denote the collection of closed terms of the language thus generated. (A term is closed if every occurrence of a variable X in it is within the scope of a μX operator). We are mostly interested in the subcollection of closed terms in which all *delays* (parameters t in the terms $P_1 \overset{t}{\rhd} P_2$, $P_1 \overset{t}{\lightning} P_2$) are integral.[3] We refer to such 'integral' terms as *programs*, the collection of which we write **TCSP**.

As an example, let us define a process *VM* intended to model a vending machine with the following behaviours: Initially, the vending machine is at any time willing to accept a coin. Once a coin has been inserted in the machine, the customer can choose between ordering a chocolate or a biscuit; however, if he fails to make his choice within 60 seconds, his money is returned and the vending machine reverts to its initial state.

We describe this vending machine below both as a Timed CSP process and as a timed automaton (cf. Section 7).

$$VM = coin.in \longrightarrow$$
$$\left(\begin{array}{c} choc \longrightarrow VM \\ \Box \\ biscuit \longrightarrow VM \end{array} \right) \overset{60}{\rhd} (coin.out \overset{!}{\longrightarrow} VM)$$



3. Denotational semantics

We present a dense-time denotational semantics for this augmented version of Timed CSP. A congruent operational semantics follows in the next section. Our denotational semantics is a combination of Reed and Roscoe's timed failures model [29, 28, 30, 35] and refusal testing [27, 20].

Designing a denotational semantics can be a subtle and difficult affair. In doing so, we were guided by the following aims:[4]

- The semantics should extend the timed failures model in a consistent and natural way.
- The semantics should be *compositional*, in that the value of a process should be entirely determined by the values of its constituent subprocesses.
- The semantics should allow us to express basic safety and liveness specifications in a natural way.

The simplest semantics to envisage is to map a program to its set of timed traces. Unfortunately, timed traces do not give rise to a compositional model [28]. They also do not extend timed failures and cannot, for instance, handle liveness specifications (as defined in Section 6).

Timed failures are timed traces together with finite unions of left-closed right-open time intervals during which certain sets of events are refused. Thus timed

[3] We could have equally well allowed rational delays, but thanks to the possibility of scaling time units, the distinction is effectively irrelevant.

[4] This 'motivating' preamble presupposes some basic knowledge of standard models of CSP and Timed CSP; the reader may wish to skip it until after having read the technical definitions of both this section and the next one.

failures only record refusals over strictly positive amounts of time. As a result, a Zeno process such as $P = a \stackrel{!}{\longrightarrow} P$ (whose only behaviours are to communicate unboundedly many a's at time 0) cannot exhibit *any* refusals in a timed failures model. Consequently, P would technically satisfy specifications such as 'The process never refuses the event b', which is nonsensical since P cannot ever perform the event b. A satisfactory framework in which processes such as P can be analyzed therefore must incorporate point (instantaneous) refusals.

Unfortunately, the naive attempt to simply augment timed failures with point refusals leads to a non-compositional model. Consider the process $Q = a \stackrel{!}{\longrightarrow} b \stackrel{!}{\longrightarrow} Q$. On any trace having at least one event, Q can record refusals of both a and b at time 0. However, if R is a renaming relation such that $a\ R\ c$ and $b\ R\ c$, then $Q[R]$ can clearly never refuse c, so that the refusals of $Q[R]$ cannot be properly calculated from those of Q.

While there are various ways to circumvent this problem, we believe the simplest is to consider a *barbed* semantics, specifically *refusal traces*, rather than timed failures. Refusal traces (defined more formally below) simply alternate refusals with performed events, in the order in which observations were made; they therefore constitute a slightly less abstract model than timed failures. Refusal traces also have very strong links with the modelling of discrete time in (untimed) CSP [23].

Some work is still required, however, in order to build a compositional model. Consider the process $a \longrightarrow b$, which can clearly refuse b prior to the occurrence of a. Should a occur at time 0, the refusal of b takes the form of a point refusal, also at time 0. The question becomes, what to do in case a is hidden, i.e., for the process $(a \longrightarrow b) \setminus a$. Timed failures and other standard models of CSP dictate that $(a \longrightarrow b) \setminus a$ should be equivalent to b. In fact, were we to record any initial refusal of $(a \longrightarrow b) \setminus a$ prior to the occurrence of τ, we would be compelled to include not only the whole of $\Sigma^{\checkmark}$, but also record that time cannot advance (because of τ-urgency), and thus wrongly conclude that $(a \longrightarrow b) \setminus a$ has a timestop.

Of the two obvious alternatives that now face us, neither leads to a compositional model, as we briefly demonstrate. The first option is to postulate that no refusals can be recorded while hidden events are on offer. Let $P_1 = a \stackrel{!}{\longrightarrow} b$ and $P_2 = ((a \stackrel{!}{\longrightarrow} b) \,\square\, c) \setminus c$. Assuming an a occurs (at time 0), P_1, unlike P_2, would have been able to record a prior refusal of b. Now consider $P = P_1 \underset{\{a,b\}}{\parallel} P_2$. Because P is required to synchronize over b's, P should refuse b whenever either of its parallel components does. However, P is clearly unable to perform an a from a stable state, and therefore cannot record a prior refusal of b.

The second alternative is to postulate that a given event can be recorded as refused at time t only if the process was unable to perform it, and either was equally unable to perform a τ-action, or chose to perform some visible event. This proposal clearly adequately handles the last situation. However, it also (rightly) makes processes b and $(c \longrightarrow b) \setminus c$ equivalent, but then distinguishes $a \,\square\, b$ and $a \,\square\, ((c \longrightarrow b) \setminus c)$: contrary to the former, the latter could record a refusal of b prior to an occurrence of a at time 0.

The simplest solution to these problems is to label a process offering τ-actions as 'unstable' at that particular time, and not to record any other information in that case; the only refusals (including the *empty* refusal!) that we are allowed to record are stable ones.[5] This idea was used in [20] to produce a compositional refusal trace model for (untimed) CSP, and later a compositional refusal trace model for discrete-time CSP [23].

It remains to decide how to handle livelock. Once again, we turn to CSP for inspiration: we treat livelock as catastrophic, and simply record on any given refusal trace the earliest time at which a process may livelock. We then disregard any subsequent behaviour of the process.

The various versions of the timed failures model are all based on complete ultrametric spaces satisfying a sizeable number of axioms, some of which are quite subtle and intricate. In general, imposing such axioms in devising a denotational semantics serves one or both of the following purposes: to achieve some form of full abstraction, in that every element of the denotational model can be realized through the syntax (cf. the failures model for CSP [31]); and to ensure the well-definedness of the various constructs, such as the existence of (unique) fixed points.

In all versions of the timed failures model, the axioms serve the second purpose only [22]. For example, the assumption of finite variability, to the effect that no process may communicate unboundedly many events in a bounded amount of time, ensures that hiding cannot introduce livelock. One finds that selectively relaxing some of the axioms is fraught with difficulties, and can easily lead to the breakdown of the semantics. On the other hand, getting rid of most of the axioms leads to a simpler and consistent model (which however is very far from full abstraction). Our main reason for choosing the second path is that, unlike the timed failures model, the model we introduce here is not only a complete ultrametric space, but also a complete partial order, which allows us to handle divergence. Thus the only axioms we require are that processes be both *downward-closed* (if some behaviour is observed, then any behaviour containing less information could also have been observed), and *divergence-closed* (once a process has entered livelock, we consider that it can exhibit any behaviour whatsoever).

We now present these ideas in more formal fashion. A *timed event* is a pair $(t, a) \in \mathbb{R}^+ \times \Sigma^{\checkmark}$. A *(timed) refusal* is a set of timed events and may also include special timed events of the form (t, time) and $(t, \bullet)$, where *time* is a symbol indicating that time cannot pass, and $\bullet$ is a symbol indicating potential instability. From now on, we abbreviate $\Sigma^{\checkmark} \cup \{\mathit{time}\}$ as $\Sigma^{\checkmark}_{\mathit{time}}$. We require in addition that refusals be time-bounded (the set of timestamps associated with a refusal's timed events must be bounded).

A *(timed) trace* is a finite sequence $\langle (t_1, a_1), (t_2, a_2), \ldots, (t_k, a_k) \rangle$ of timed events, with the timestamps (denoting global, or absolute, time) non-decreasing.

[5] A rather intricate discrete-time model in which unstable refusals are recorded is presented in [23]; while this model is consistent and provides detailed information about processes, it appears to be of limited practical utility.

A *(timed) refusal trace* is a finite sequence $\langle \aleph_0, (t_1, a_1), \aleph_1, (t_2, a_2), \ldots, (t_k, a_k), \aleph_k \rangle$ (with $k \geqslant 0$), where each $\aleph_i$ is a refusal and each (t_i, a_i) is a timed event, subject to the following conditions:

(1) Time is non-decreasing: the t_i's are non-decreasing, and moreover, for all $1 \leqslant i \leqslant k$, whenever $(t, \alpha) \in \aleph_{i-1}$, then $t \leqslant t_i$, and whenever $(t, \alpha) \in \aleph_i$, then $t_i \leqslant t$.

(2) An event cannot be refused and then observed at the same time: for all $1 \leqslant i \leqslant k$, $(t_i, a_i) \notin \aleph_{i-1}$.

(3) If time is refused, then time stops: for all $0 \leqslant i \leqslant k$, if $(t, \mathit{time}) \in \aleph_i$, then all timed events in $\aleph_i$ have timestamps no greater than t, and moreover if $i < k$ then $t_{i+1} = t$.

(4) No refusals are recorded while a process is potentially unstable: for all $0 \leqslant i \leqslant k$, if $\{(t, \bullet), (t, \alpha)\} \subseteq \aleph_i$, then $\alpha = \bullet$.

(5) Once a process terminates, only time elapses: if $i < k$, then $a_i \neq \checkmark$, and if $a_k = \checkmark$, then $(t, \mathit{time}) \notin \aleph_k$ for any value of t.

A *(timed) refusal trace with divergence* (*rtd* for short) is a pair (T, z), where T is a refusal trace and $z \in \mathbb{R}^+ \cup \{\infty\}$ is greater than or equal to any of the timestamps associated with events (observed or refused) in T. The set of all rtd's is written *RTD*.

We interpret an rtd (T, z) as a summary of an experimenter's interaction with a process, in which events that were refused were recorded in the various refusals of T, whereas events that were observed were recorded in between refusals. Moreover, we imagine that processes come equipped with two lights, respectively labelled 'time' and 'unstable'. When lit, the 'time' light indicates that time cannot advance; a refusal of *time* may therefore be recorded. The 'unstable' light goes on whenever the process has one or more τ-actions on offer. When this light is on, the only refusal we are allowed to record at that time is $\bullet$, irrespective of any other or subsequent observations we make.

However, we do not necessarily observe and record all possible refusals. We may choose to observe the process at any instant in time, or for any continuous period of time, and record our observations as indicated above. We are nonetheless required to conservatively record a $\bullet$-refusal whenever we did not actually observe the process at a particular time. Note that a process may go through a sequence of several states at the same instant, any of which could lead to a specific refusal. For example, if observed at time 0, the process $(a \longrightarrow b) \setminus a$ may have either $\bullet$ or $\Sigma^{\checkmark} - \{b\}$ as (maximal) refusals.

Lastly, if the 'unstable' light remains lit for longer than an instant, we record a livelock by setting z equal to the earliest time at which we began observing divergence. Otherwise we let $z = \infty$.

As discussed earlier, we treat livelock as catastrophic, and do not attempt to record anything 'past' it. To simplify our modelling task, we consider that a process which has entered livelock is capable of any behaviour whatsoever. A process is *livelock-free* if every one of its rtd's has a z-value of ∞. In this work, all processes that we consider are livelock-free; in general, one can invoke certain static analysis techniques to guarantee that timed processes are livelock-free [25].

Let $T = \langle \aleph_0, (t_1, a_1), \ldots, \aleph_k \rangle$ and $T' = \langle \aleph'_0, (t'_1, a'_1), \ldots, \aleph'_l \rangle$. We define their 'glueing' $T \frown T' \mathrel{\widehat{=}} \langle \aleph_0, (t_1, a_1), \ldots, (t_k, \aleph_k), \aleph_k \cup \aleph'_0, (t'_1, a'_1), \ldots, \aleph'_l \rangle$.

If T is a refusal trace and $t \in \mathbb{R}^+$, we let $T + t$ be the refusal trace in which all timed events of T (observed and refused) have had their timestamps increased by t. We also let $T + \infty \mathrel{\widehat{=}} \langle \emptyset \rangle$.

Let $P \subseteq RTD$ be a set of rtd's. We define the *divergence-closure* of P to be $\overline{P} \mathrel{\widehat{=}} \{(T \frown (T' + z), z' + z) \mid (T, z) \in P \wedge (T', z') \in RTD\} \cap RTD$. We then say that a set of rtd's P is *divergence-closed* if $P = \overline{P}$.

Let $\aleph, \aleph'$ be refusals. Define $\aleph \prec \aleph'$ if $\aleph \cap (\mathbb{R}^+ \times \{\bullet\}) \supseteq \aleph' \cap (\mathbb{R}^+ \times \{\bullet\})$ and $\aleph \subseteq \aleph' \cup (\mathbb{R}^+ \times \{\bullet\})$. Now let $(T, z) = (\langle \aleph_0, (t_1, a_1), \ldots, \aleph_k \rangle, z)$ and $(T', z') = (\langle \aleph'_0, (t'_1, a'_1), \ldots, \aleph'_l \rangle, z')$ be rtd's. We write $(T, z) \prec (T', z')$ (representing the fact that (T', z') contains at least as much information as (T, z)), when either of the following two conditions are met:

$$z = \infty \wedge k \leqslant l \wedge \aleph_0 \prec \aleph'_0 \wedge \forall (1 \leqslant i \leqslant k) \centerdot a_i = a'_i \wedge \aleph_i \prec \aleph'_i \ \text{, or}$$
$$z = z' \wedge k = l \wedge \aleph_0 \prec \aleph'_0 \wedge \forall (1 \leqslant i \leqslant k) \centerdot a_i = a'_i \wedge \aleph_i \prec \aleph'_i \ .$$

For $P \subseteq RTD$, we let $\downarrow P \mathrel{\widehat{=}} \{(T, z) \in RTD \mid \exists (T', z') \in P \centerdot (T, z) \prec (T', z')\}$. We then say that a set of rtd's P is *downward-closed* if $P = \downarrow P$.

D 1. *The timed refusal traces with divergence model $\mathcal{RTD}$ is the set consisting of all downward-closed and divergence-closed $P \subseteq RTD$.*

T 1. *$\mathcal{RTD}$ is a complete partial order (in fact a complete lattice) under reverse set inclusion, denoted $\sqsubseteq$. Its least element is RTD. Moreover, every n-ary Timed CSP operator other than recursion can be interpreted as a continuous function $\mathcal{RTD}^n \longrightarrow \mathcal{RTD}$. Recursion is interpreted as the least fixed point operator on $\mathcal{RTD}$. These interpretations define an algebra homomorphism (i.e., a compositional map) $\mathcal{R}_{\mathbb{R}}[\![\cdot]\!]$:* **TCSP** *$\longrightarrow \mathcal{RTD}$. This map can also be defined through the operational semantics (presented next).*

Theorem 1 is similar to corresponding results in the context of timed failures [28], discrete-time refusal traces [23], and (untimed) CSP [20, 31], and can be established using the same standard proof techniques.

For P a Timed CSP program, $\mathcal{R}_{\mathbb{R}}[\![P]\!]$ represents the set of refusal traces with divergence that an experimenter can observe while interacting with P. For example, the program $a \overset{2}{\rhd} b \overset{!}{\longrightarrow} a$ has, among others, the following refusal trace over alphabet $\Sigma = \{a, b\}$: $\langle \aleph_0, (2, b), \aleph_1, (2.7, a), \aleph_2 \rangle$, where $\aleph_0 = [0, 2) \times \{b\} \cup \{2\} \times \{a, \mathit{time}\}$, $\aleph_1 = [2, 2.7] \times \{b\}$, and $\aleph_2 = [2.7, 3.4] \times \{a, b\}$. Note that the refusal of $(2, \mathit{time})$ in $\aleph_0$ arises from the presence of the signal b.

Let us briefly once again contrast our framework with the standard timed failures model. The refusal components of timed failures are defined to be finite unions of *refusal tokens*: refusals of the form $[t, t') \times A$. As it turns out, for **TCSP** programs, our (maximal) refusals also consist of finite unions of refusal tokens, together with finitely many *point refusals*—refusals of the form $\{t\} \times A$. As discussed earlier, these enable us to appropriately model signals. The divergence component (z) of rtd's allows us to model livelock, while the refusal event *time* is not only useful for modelling signals, but also for detecting timestops. Finally, the unstable refusal event $\bullet$ lets us draw the distinction between (potential) instability and stability, and in doing so allows for compositional modelling.

Timed failures do not form a complete partial order, among other reasons because the axiom of finite variability precludes the existence of a least element. Consequently, the timed failures semantics cannot model recursion in full generality (i.e., in the presence of Zeno or divergent processes).

One can also compositionally define $\mathcal{R}_{\mathbb{Z}}[\![P]\!]$, the set of integral-time rtd's of a process P. A refusal trace is *integral-time* if each of its events has an integral timestamp, and if each of its refusals can be written as a union of refusal tokens and point refusals, all with integral endpoints.

Finally, we let $\mathcal{T}_{\mathbb{R}}[\![P]\!]$ and $\mathcal{T}_{\mathbb{Z}}[\![P]\!]$ respectively stand for the sets of dense-time and integral-time timed traces of P. (Note that these cannot be defined compositionally; however, they can be extracted straightforwardly from $\mathcal{R}_{\mathbb{R}}[\![P]\!]$ or from the operational semantics.)

4. Operational semantics

The contents and style of this section are similar to [34]. We present a collection of inference rules which allow us to assign to any closed term in *TCSP* a set of dense-time *executions*.

We list a few notational conventions: a and b stand for visible events, i.e., belong to $\Sigma^{\checkmark}$. $A \subseteq \Sigma$ and $A^{\checkmark} = A \cup \{\checkmark\}$. γ can be a visible event or a silent one ($\gamma \in \Sigma^{\checkmark} \cup \{\tau\}$). $P \xrightarrow{\gamma} P'$ means that the closed term P can perform an immediate and instantaneous γ-transition, and subsequently become P' (communicating γ in the process if γ is a visible event). $P \stackrel{\gamma}{\not\longrightarrow}$ means that P cannot possibly do a γ at that particular time. $P \stackrel{t}{\rightsquigarrow} P'$ means that P can become P' simply by virtue of letting t units of time elapse, where $t \in \mathbb{R}^+$. $P \rightsquigarrow$ means that P can let some strictly positive amount of time elapse, whereas $P \not\rightsquigarrow$ stands for the opposite. In what follows, $u \in \mathbb{R}^+$. If P is a term with a single free variable X and Q is a closed term, $[Q/X]P$ represents the closed term P with Q substituted for every free occurrence of X.

The rules are as follows.

$$\frac{}{STOP \stackrel{t}{\rightsquigarrow} STOP}$$

$$\frac{}{(a \longrightarrow P) \stackrel{t}{\rightsquigarrow} (a \longrightarrow P)} \qquad \frac{}{(a \longrightarrow P) \xrightarrow{a} P}$$

$$\frac{}{(a \xrightarrow{!} P) \xrightarrow{a} P}$$

$$\frac{}{SKIP \stackrel{t}{\rightsquigarrow} SKIP} \qquad \frac{}{SKIP \xrightarrow{\checkmark} STOP}$$

$$\frac{}{\mathit{RANDOM} \xrightarrow{\tau} \mathit{WAIT}\ u}$$

$$\frac{}{\mathit{WAIT}\ u \stackrel{t}{\rightsquigarrow} \mathit{WAIT}\ (u-t)}\ [\, t \leqslant u \,] \qquad \frac{}{\mathit{WAIT}\ 0 \xrightarrow{\tau} \mathit{SKIP}}$$

$$\frac{P_1 \stackrel{t}{\rightsquigarrow} P_1'}{P_1 \stackrel{u}{\rhd} P_2 \stackrel{t}{\rightsquigarrow} P_1' \stackrel{u-t}{\rhd} P_2}\ [\, t \leqslant u \,]$$

$$\frac{}{P_1 \stackrel{0}{\rhd} P_2 \xrightarrow{\tau} P_2} \qquad \frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \stackrel{u}{\rhd} P_2 \xrightarrow{\tau} P_1' \stackrel{u}{\rhd} P_2} \qquad \frac{P_1 \xrightarrow{a} P_1'}{P_1 \stackrel{u}{\rhd} P_2 \xrightarrow{a} P_1'}$$

$$\frac{P_1 \stackrel{t}{\rightsquigarrow} P_1'}{P_1 \stackrel{u}{\lightning} P_2 \stackrel{t}{\rightsquigarrow} P_1' \stackrel{u-t}{\lightning} P_2}\ [\, t \leqslant u \,]$$

$$\frac{}{P_1 \stackrel{0}{\lightning} P_2 \xrightarrow{\tau} P_2} \qquad \frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \stackrel{u}{\lightning} P_2 \xrightarrow{\checkmark} P_1'} \qquad \frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \stackrel{u}{\lightning} P_2 \xrightarrow{\gamma} P_1' \stackrel{u}{\lightning} P_2}\ [\, \gamma \neq \checkmark \,]$$

$$\frac{P_1 \stackrel{t}{\rightsquigarrow} P_1' \quad P_2 \stackrel{t}{\rightsquigarrow} P_2'}{P_1 \Box P_2 \stackrel{t}{\rightsquigarrow} P_1' \Box P_2'}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \Box P_2 \xrightarrow{\tau} P_1' \Box P_2} \qquad \frac{P_2 \xrightarrow{\tau} P_2'}{P_1 \Box P_2 \xrightarrow{\tau} P_1 \Box P_2'}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \Box P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \Box P_2 \xrightarrow{a} P_2'}$$

$$\frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_2}$$

$$\frac{P_1 \stackrel{t}{\rightsquigarrow} P_1' \quad P_2 \stackrel{t}{\rightsquigarrow} P_2'}{P_1 \underset{A}{\parallel} P_2 \stackrel{t}{\rightsquigarrow} P_1' \underset{A}{\parallel} P_2'}$$

$$\frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{\gamma} P_1' \underset{A}{\parallel} P_2}\ [\, \gamma \notin A^{\checkmark} \,] \qquad \frac{P_2 \xrightarrow{\gamma} P_2'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{\gamma} P_1 \underset{A}{\parallel} P_2'}\ [\, \gamma \notin A^{\checkmark} \,]$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{a} P_1' \underset{A}{\parallel} P_2'}\ [\, a \in A^{\checkmark} \,]$$

$$\frac{P_1 \stackrel{t}{\rightsquigarrow} P_1' \quad P_1 \stackrel{\checkmark}{\not\longrightarrow}}{P_1 \fatsemi P_2 \stackrel{t}{\rightsquigarrow} P_1' \fatsemi P_2} \qquad \frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \fatsemi P_2 \xrightarrow{\tau} P_2} \qquad \frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \fatsemi P_2 \xrightarrow{\gamma} P_1' \fatsemi P_2}\ [\, \gamma \neq \checkmark \,]$$

$$\frac{P \overset{t}{\rightsquigarrow} P' \quad \forall\, a \in A \,.\, P \overset{a}{\not\longrightarrow}}{P \setminus A \overset{t}{\rightsquigarrow} P' \setminus A}$$

$$\frac{P \overset{a}{\longrightarrow} P'}{P \setminus A \overset{\tau}{\longrightarrow} P' \setminus A}\,[\, a \in A\,] \qquad \frac{P \overset{\gamma}{\longrightarrow} P'}{P \setminus A \overset{\gamma}{\longrightarrow} P' \setminus A}\,[\, \gamma \notin A\,]$$

$$\frac{P \overset{t}{\rightsquigarrow} P'}{P[R] \overset{t}{\rightsquigarrow} P'[R]} \qquad \frac{P \overset{\tau}{\longrightarrow} P'}{P[R] \overset{\tau}{\longrightarrow} P'[R]} \qquad \frac{P \overset{a}{\longrightarrow} P'}{P[R] \overset{b}{\longrightarrow} P'[R]}\,[\, a\, R\, b\,]$$

$$\frac{}{\mu X \,.\, P \overset{\tau}{\longrightarrow} [(\mu X \,.\, P)/X]P}$$

$$\frac{}{DIV \overset{\tau}{\longrightarrow} DIV} \;.$$

Note that there are no rules for *TIMESTOP*.

The reader may have noticed that two of our rules (dealing with termination and hiding) incorporate *negative premisses* ($P_1 \overset{\checkmark}{\not\longrightarrow}$ and $P \overset{a}{\not\longrightarrow}$), which could potentially yield an inconsistent definition. This does not occur, for the following reason: notice that the $\overset{\gamma}{\longrightarrow}$ relation can be defined, *independently of the* $\overset{t}{\rightsquigarrow}$ *relation*, as the smallest relation satisfying the relevant subset of rules, since no negative premisses are involved in its definition. Once the $\overset{\gamma}{\longrightarrow}$ relation has been defined, the $\overset{t}{\rightsquigarrow}$ relation can then itself be defined. Since the negative premisses are all phrased in terms of the previously defined (and fixed) $\overset{\gamma}{\longrightarrow}$ relation, they do not pose any problem.

We now list a number of definitions and results concerning the operational semantics which we will require later on.

For $P \in TCSP$, we define an *execution* of P to be a sequence $e = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n$, where $P_0 = P$ and each subsequence $P_i \overset{z_{i+1}}{\longmapsto} P_{i+1}$ of e is either a transition $P_i \overset{\gamma}{\longrightarrow} P_{i+1}$ (with $z_{i+1} = \gamma$), or an evolution $P_i \overset{t}{\rightsquigarrow} P_{i+1}$ (with $z_{i+1} = t$). In addition, every such transition or evolution must be validly allowed by the operational inference rules listed above. The set of executions of P is written $\mathsf{exec}(P)$.

Given an execution e as above and a non-negative integer $k \leqslant n$, we define the *prefix* $e(k)$ of e to be the execution $P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_k}{\longmapsto} P_k$.

Given two Timed CSP terms $P, Q \in TCSP$, write $P \sim Q$ if P and Q are syntactically identical except possibly for the various values of the delays in their respective subterms. Thus $(WAIT\ 0.5 \ ||| \ STOP) \sim (WAIT\ 0 \ ||| \ STOP)$, but $(STOP \ \Box \ STOP) \nsim STOP$ and $WAIT\ 0 \nsim SKIP$. $\sim$ is clearly an equivalence relation.

The next two propositions are easily established by structural induction.

P 1. τ-*urgency: time cannot evolve while hidden events are on offer— for any* $P \in TCSP$, *if* $P \overset{\tau}{\longrightarrow}$ *then* $P \not\rightsquigarrow$.

P 2. *Persistency: for any $P, Q \in TCSP$, if $P \overset{t}{\rightsquigarrow} Q$ then $P \sim Q$. Moreover, whenever $P \sim Q$, then for any visible event $a \in \Sigma^{\checkmark}$, $P \xrightarrow{a}$ if and only if $Q \xrightarrow{a}$.*

Note that we may have $P \sim Q$ and $P \xrightarrow{\tau}$ whereas $Q \overset{\tau}{\not\longrightarrow}$.

For e an execution, define its *duration* $\mathsf{dur}(e)$ to be the sum of the durations of its evolutions.

For $P \in TCSP$, define the set of events immediately refused by P, $\mathsf{ref}(P) \subseteq \Sigma^{\checkmark}_{time} \cup \{\bullet\}$, as follows: if $P \xrightarrow{\tau}$, then $\mathsf{ref}(P) \mathrel{\widehat{=}} \{\bullet\}$. Otherwise, for $a \in \Sigma^{\checkmark}$, $a \in \mathsf{ref}(P) \Leftrightarrow P \overset{a}{\not\longrightarrow}$, and $time \in \mathsf{ref}(P) \Leftrightarrow P \not\rightsquigarrow$.

We say that P *diverges* if P has an infinite evolution-less execution whose transitions are exclusively τ's.

Given an execution e of some program P, we produce an associated canonical refusal trace with divergence $\mathsf{rtd}(e)$ (the largest possible given the execution e), defined inductively on e as follows.

$$\mathsf{rtd}(P) \mathrel{\widehat{=}} \begin{cases} (\langle\emptyset\rangle, 0) & \text{if } P \text{ diverges} \\ (\langle\{0\} \times \mathsf{ref}(P)\rangle, \infty) & \text{otherwise} \end{cases}$$

$$\mathsf{rtd}(P \xrightarrow{\tau} {}^\frown e) \mathrel{\widehat{=}} \mathsf{rtd}(e)$$

$$\mathsf{rtd}(P \xrightarrow{a} {}^\frown e) \mathrel{\widehat{=}} (\langle\{0\} \times \mathsf{ref}(P), (0, a)\rangle^\frown T, z) \text{ if } \mathsf{rtd}(e) = (T, z)$$

$$\mathsf{rtd}(P \overset{t}{\rightsquigarrow} {}^\frown e) \mathrel{\widehat{=}} (\langle[0, t) \times \mathsf{ref}(P)\rangle \curvearrowright (T + t), z + t) \text{ if } \mathsf{rtd}(e) = (T, z) \ .$$

(The operator $^\frown$ denotes sequence concatenation.)

The congruence theorem reads:

T 2. *For any $P \in$ **TCSP**, $\mathcal{R}_{\mathbb{R}}[\![P]\!] = \overline{\downarrow \mathsf{rtd}(\mathsf{exec}(P))}$.*

The proof techniques employed in [34, 23] to establish similar congruence results can also be used to prove Theorem 2.

5. Digitization

Digitization techniques were first introduced in [15], and later extended in the context of Timed CSP from traces to timed failures in [23, 24]. We review the main points, adapted to the present framework.

Let $t \in \mathbb{R}^+$, and let $0 \leqslant \delta \leqslant 1$ be a real number. Decompose t into its integral and fractional parts, thus: $t = \lfloor t \rfloor + fract(t)$. If $fract(t) \leqslant \delta$, let $[t]_\delta \mathrel{\widehat{=}} \lfloor t \rfloor$, otherwise let $[t]_\delta \mathrel{\widehat{=}} \lceil t \rceil$. The $[\cdot]_\delta$ operator therefore shifts the value of a real number t to the preceding or following integer, depending on whether the fractional part of t is less than or equal to δ or not.

We can extend $[\cdot]_\delta$ to timed traces by pointwise application to the timestamps of the trace's events. We then further extend $[\cdot]_\delta$ to sets of traces in the usual way.[6]

[6] Digitizing refusals is also possible, but leads to complications due to the presence of point refusals.

D 2. *A set P of timed traces is* closed under digitization *if, for any* $0 \leqslant \delta \leqslant 1$, $[P]_\delta \subseteq P$.

A set S of timed traces is closed under inverse digitization *if, whenever a trace s is such that* $[s]_\delta \in S$ *for all* $0 \leqslant \delta \leqslant 1$, *then* $s \in S$.

We extend this definition to Timed CSP programs P and S by applying it to $\mathcal{T}_{\mathbb{R}}[\![P]\!]$ *and* $\mathcal{T}_{\mathbb{R}}[\![S]\!]$.

If P is a set of timed traces, we let $\mathbb{Z}(P)$ stand for the subset of integral-time timed traces of P.

The central verification result is as follows:

T 3. *Let P be a set of timed traces closed under digitization, and let S be a set of timed traces closed under inverse digitization. Then* $P \subseteq S$ *if and only if* $\mathbb{Z}(P) \subseteq \mathbb{Z}(S)$.

The proof is straightforward [15].

As regards Timed CSP, we have:

P 3. *Any Timed CSP program* $P \in \mathbf{TCSP}$ *is closed under digitization.*

Proposition 3 follows directly from Lemma 1 (below) and Theorem 2.

The Digitization Lemma reads as follows:

L 1. *Let* $P \in \mathbf{TCSP}$, *and let* $e = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n \in \mathsf{exec}(P)$. *For any* $0 \leqslant \delta \leqslant 1$, *there exists an execution* $[e]_\delta = P'_0 \overset{z'_1}{\longmapsto} P'_1 \overset{z'_2}{\longmapsto} \ldots \overset{z'_n}{\longmapsto} P'_n \in \mathsf{exec}(P)$ *with the following properties:*

(1) *The transitions and evolutions of e and* $[e]_\delta$ *are in natural one-to-one correspondence. More precisely, whenever* $P_i \overset{z_{i+1}}{\longmapsto} P_{i+1}$ *in e is a transition, then so is* $P'_i \overset{z'_{i+1}}{\longmapsto} P'_{i+1}$ *in* $[e]_\delta$, *and moreover* $z'_{i+1} = z_{i+1}$. *On the other hand, whenever* $P_i \overset{z_{i+1}}{\longmapsto} P_{i+1}$ *in e is an evolution, then so is* $P'_i \overset{z'_{i+1}}{\longmapsto} P'_{i+1}$ *in* $[e]_\delta$, *with* $|z_{i+1} - z'_{i+1}| < 1$.

(2) *All evolutions in* $[e]_\delta$ *have integral duration.*

(3) $P'_0 = P_0 = P$; *in addition, for all* $0 \leqslant i \leqslant n$, $P'_i \in \mathbf{TCSP}$ *and* $P'_i \sim P_i$.

(4) *For any prefix* $e(k)$ *of e, we have* $\mathsf{dur}([e]_\delta(k)) = [\mathsf{dur}(e(k))]_\delta$.

(5) *Lastly, for any prefix* $e(k)$ *of e, the* $[\cdot]_\delta$ *operator is a function of the prefix only:* $[e(k)]_\delta = [e]_\delta(k)$.

The proof proceeds by structural induction on P. The details are carefully laid out in [23], in the context of standard Timed CSP. It is straightforward to extend that proof to handle signals (dealt with the same way as τ-events) as well as the process *RANDOM*.

We also record the following result, which will be useful later on:

A detailed exposition of the application of digitization to timed failures can be found in the references mentioned above. However, since the primary focus of the current paper lies elsewhere, we shall be content with using somewhat simpler (if slightly more ad-hoc) techniques when it comes to refusals.

P 4. *Let* $P \in \mathbf{TCSP}$, *and let* $e = P_0 \overset{z_1}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n$ *be an execution of* P. *Let* $[e]_{fract(\mathsf{dur}(e))} = P_0' \overset{z_1'}{\longmapsto} \ldots \overset{z_n'}{\longmapsto} P_n'$ *be the digitization of* e *by the fractional part of its duration. Then* $\mathsf{ref}(P_n) = \mathsf{ref}(P_n')$.

P . Let e be as above, and suppose that $\mathsf{ref}(P_n) \neq \mathsf{ref}(P_n')$. Since $P_n \sim P_n'$, by Proposition 2, either $P_n \overset{\tau}{\not\longrightarrow}$ and $P_n' \overset{\tau}{\longrightarrow}$, or vice-versa.

Note that if $P_n \overset{\tau}{\longrightarrow} R$ (for some $R \in TCSP$), then by applying the $[\cdot]_{fract(\mathsf{dur}(e))}$ operator to the extended execution $e^\frown(\overset{\tau}{\longrightarrow} R)$, we immediately conclude that $P_n' \overset{\tau}{\longrightarrow}$ as well.

Thus suppose that $P_n \overset{\tau}{\not\longrightarrow}$ and $P_n' \overset{\tau}{\longrightarrow}$. Define a syntactic function $!^{-1} : TCSP \longrightarrow TCSP$ which, given a term Q, returns a term $!^{-1}(Q)$ that is identical to Q except that all signalling subterms $a \overset{!}{\longrightarrow} \ldots$ of Q are simply replaced by $a \longrightarrow \ldots$ in $!^{-1}(Q)$. $!^{-1}$ also replaces subterms *TIMESTOP* by *STOP*. This function extends to executions in the obvious way. One easily shows that $!^{-1}(Q) \not\rightsquigarrow$ if and only if $!^{-1}(Q) \overset{\tau}{\longrightarrow}$. Observe moreover that, for any visible or hidden event $\gamma \in \Sigma^{\checkmark} \cup \{\tau\}$, $Q \overset{\gamma}{\longrightarrow}$ if and only if $!^{-1}(Q) \overset{\gamma}{\longrightarrow}$.

Consider the execution $!^{-1}(e)$ of $!^{-1}(P)$. Since $!^{-1}(P_n) \overset{\tau}{\not\longrightarrow}$, there exists $t > 0$ such that $!^{-1}(P_n) \overset{t}{\rightsquigarrow} R$, for some $R \in TCSP$. Digitizing $!^{-1}(e)$ extended by this last evolution, we get $[!^{-1}(e)^\frown(\overset{t}{\rightsquigarrow} R)]_{fract(\mathsf{dur}(e))} = !^{-1}(P_0' \overset{z_1'}{\longmapsto} \ldots \overset{z_n'}{\longmapsto} P_n') \overset{t'}{\rightsquigarrow} R'$, for some $t' > 0$ and $R' \in \mathbf{TCSP}$. Proposition 1 therefore implies that $!^{-1}(P_n') \overset{\tau}{\not\longrightarrow}$, from which we derive the contradiction $P_n' \overset{\tau}{\not\longrightarrow}$, as required. □

D 3. *Let* $P \in \mathbf{TCSP}$ *be a Timed CSP program. We say that* P *is* finite-state *if the collection of* $\sim$*-equivalence classes of programs reachable from* P *is finite.*

In other words, P is finite-state if, according to the operational semantics, it can give rise to only finitely many $\sim$-distinct other programs.

A very useful property of finite-state programs is that they have finite integral labelled transition systems, making them amenable to automated analysis through digitization and model checking.

P 5. *Let* $P \in \mathbf{TCSP}$ *be a finite-state program. Let* P' *be the program obtained from* P *by replacing every occurrence of RANDOM in* P *by the term* $(\mu X \centerdot SKIP \overset{0}{\rhd} WAIT\ 1 \,;\, X)$. *Then the set of programs reachable from* P' *through integral-time executions (i.e., executions all of whose evolutions have integral durations) is finite.*

(The converse also holds.)

Note that P and P' have the same integral denotational value: $\mathcal{R}_{\mathbb{Z}}[\![P]\!] = \mathcal{R}_{\mathbb{Z}}[\![P']\!]$, since $\mathcal{R}_{\mathbb{Z}}[\![RANDOM]\!] = \mathcal{R}_{\mathbb{Z}}[\![\mu X \centerdot SKIP \overset{0}{\rhd} WAIT\ 1 \,;\, X]\!]$.

P . Follows directly from Lemma 1. □

6. Specifications as refinements, and verification

We consider the questions of expressing specifications on processes as refinements (reverse inclusion of sets of behaviours), and of verifying such specifications. We are interested both in *trace* refinements (capturing safety properties: 'nothing bad happens') and *refusal trace* refinements (capturing both safety and liveness: 'good things are not prevented from happening').

Note that liveness is for us a branching-time concept, different from Alpern and Schneider's related linear-time definition [1]. The latter can be paraphrased as 'something good must eventually happen': messages are eventually delivered, the printer is eventually online, etc. In our case, examples of liveness include 'the eject button is always enabled once the aircraft is in the air', 'the network is deadlock-free (or timestop-free)', 'the nuclear warheads are permanently launch-ready', etc. Observe that there is no requirement that the live behaviour in question actually ever take place; indeed, the security provided by, say, a nuclear deterrent, lies precisely in the fact that it need *not* ever be used.

Real-time specifications are usually expressed in some temporal logic, such as MTL (linear-time) or TCTL (branching-time) [6]. The question of delineating the exact expressive power of refinement with respect to such logics is studied in [19] in the untimed case and shown to be a subtle problem. The addition of time naturally compounds the difficulties. A comprehensive treatment of the question is therefore a challenging topic for further work; nonetheless, we show here that many, if not most, interesting real-time properties can indeed be captured as Timed CSP refinements.

An implementation $P \in \mathbf{TCSP}$ meets a specification $S \in \mathbf{TCSP}$ if all the behaviours of P are also behaviours of S. This leads to four possible definitions of satisfaction, according to whether the behaviours considered are timed refusal traces or timed traces, and according to whether time is dense or discrete (integral):

$$\begin{aligned}
P \vDash^{\mathcal{R}}_{\mathbb{R}} S &\Leftrightarrow \mathcal{R}_{\mathbb{R}}[\![P]\!] \subseteq \mathcal{R}_{\mathbb{R}}[\![S]\!] \\
P \vDash^{\mathcal{R}}_{\mathbb{Z}} S &\Leftrightarrow \mathcal{R}_{\mathbb{Z}}[\![P]\!] \subseteq \mathcal{R}_{\mathbb{Z}}[\![S]\!] \\
P \vDash^{\mathcal{T}}_{\mathbb{R}} S &\Leftrightarrow \mathcal{T}_{\mathbb{R}}[\![P]\!] \subseteq \mathcal{T}_{\mathbb{R}}[\![S]\!] \\
P \vDash^{\mathcal{T}}_{\mathbb{Z}} S &\Leftrightarrow \mathcal{T}_{\mathbb{Z}}[\![P]\!] \subseteq \mathcal{T}_{\mathbb{Z}}[\![S]\!] \ .
\end{aligned}$$

We present below three paradigmatic linear-time safety specifications (safe reachability, bounded invariance, and bounded response), and briefly describe how they can be expressed as Timed CSP processes. We also present two paradigmatic branching-time liveness specifications (constant availability and timestop-freedom) and likewise show how they are captured by Timed CSP processes.

Three of the specifications are given their corresponding MTL formulas as names, but no knowledge of MTL is required or assumed. For simplicity, we have ignored the possibility of global successful termination (communication of $\checkmark$'s).

- *Safe reachability ($\Box\neg a$)*: 'The event a is never performed.' According to [11], this is the most common specification on timed systems, since "most properties [on timed systems] can be encoded as exceptions [the event a in this case]". This is a trace specification which is captured by the process $RUN_{\Sigma-\{a\}}$. Here

$$RUN_B = \Box\{b \longrightarrow RUN_B \mid b \in B\}$$

can perform any trace containing only events in B.

- *Constant availability ($AVAIL_{\{a\}}$)*: 'The event a is never refused.' The process $AVAIL_{\{a\}}$ captures this refusal trace liveness specification. Here

$$AVAIL_B = \Box\{b \longrightarrow AVAIL_B \mid b \in B\} \;\Box\; RANDOM \,;\, (\bigsqcap\{c \xrightarrow{!} AVAIL_B \mid c \in \Sigma\} \sqcap TIMESTOP)$$

  is a livelock-free process which can perform any trace and refuse time as well as any set of events outside of B.

- *Timestop-freedom (TSF)*: 'The process never exhibits timestops.' In other words, the process never reaches a point where the whole of $\Sigma^{\checkmark}_{time}$ is refused. The process

$$TSF = RANDOM \,;\, \bigsqcap\{a \xrightarrow{!} TSF \mid a \in \Sigma\}$$

  captures this refusal trace liveness specification. TSF is the most nondeterministic process which has no timestops or livelocks.

- *Bounded invariance ($\Box(a \Rightarrow \Box_I \neg b)$)*: 'Whenever the event a occurs, the event b is prevented from occurring during the time interval I, as measured from the time of occurrence of a.' Here $I = (k, k')$ is an open interval of length at least two with integral (or infinite) endpoints. Bounded invariance and bounded response (next) are listed in [15] as the two specifications most commonly encountered in practice; note that safe reachability is essentially a special case of bounded invariance.

  We capture this trace specification by a process containing $2\lceil\frac{k}{k'-k}\rceil+2$ parallel processes. All but one of these are 'alarm clocks': whenever an a occurs, two alarm clocks are set up, one to ring in k time units, indicating that b's should be disabled, the other to ring in k' time units, to end the prohibition on b's. Now should a second a occur within $k' - k$ time units (a single clock is used to keep track of this time period), the second of the two alarm clocks just described is simply reset to ring k' time units in the future. A single discrete controller easily manages all these clocks. Note that the 'alarm rings' are internal, i.e., globally hidden.

- *(Strong) bounded response ($\Box(a \Rightarrow \Diamond_J b)$)*: 'Whenever the event a occurs, the event b must occur during the time interval J, as measured from the time of occurrence of a.' Here $J = [k, k']$ is a closed interval with integral endpoints and $k < k'$ or $k = k' = 0$.

  In general the most nondeterministic process satisfying a bounded response property will be infinite-state (require infinitely many clocks); however we can define a finite-state process which captures the *integral* behaviours of this trace specification. Such a process consists of a discrete controller along with $k' + 1$ clocks. Again, for a given occurrence of the event a, an alarm clock is set to ring after k time units have passed. This indicates the beginning of the period during which the event b must occur. Having rung, the clock is reset to ring $k' - k$ time units later, at the very end of the period in question. b's are constantly on offer, and as soon as one occurs, the monitoring of b's is disengaged. Otherwise, a b is signalled (and thus must happen on the spot) when the second alarm goes off. The fact that this process only captures the integral behaviours of the corresponding bounded response property is sufficient for verification purposes, thanks to digitization:

P 6. *The specifications of safe reachability, bounded invariance, and bounded response are closed under inverse digitization.*

We refer the reader to [15] for the proof.

T 4. *Let $P \in$ **TCSP** be a Timed CSP process. Then*

$$P \vDash^{\mathcal{T}}_{\mathbb{R}} \Box\neg a \Leftrightarrow P \vDash^{\mathcal{T}}_{\mathbb{Z}} \Box\neg a$$
$$P \vDash^{\mathcal{R}}_{\mathbb{R}} AVAIL_{\{a\}} \Leftrightarrow P \vDash^{\mathcal{R}}_{\mathbb{Z}} AVAIL_{\{a\}}$$
$$P \vDash^{\mathcal{R}}_{\mathbb{R}} TSF \Leftrightarrow P \vDash^{\mathcal{R}}_{\mathbb{Z}} TSF$$
$$P \vDash^{\mathcal{T}}_{\mathbb{R}} \Box(a \Rightarrow \Box_I \neg b) \Leftrightarrow P \vDash^{\mathcal{T}}_{\mathbb{Z}} \Box(a \Rightarrow \Box_I \neg b)$$
$$P \vDash^{\mathcal{T}}_{\mathbb{R}} \Box(a \Rightarrow \Diamond_J b) \Leftrightarrow P \vDash^{\mathcal{T}}_{\mathbb{Z}} \Box(a \Rightarrow \Diamond_J b) \ .$$

In other words, for each of the specifications S considered, P satisfies S over dense time if and only if P satisfies S over discrete time.

Note that the assertions on the right-hand side are all discrete and finite-state. They can be verified (under certain conditions) on the model checker FDR via encoding into (untimed) CSP; see [23, 24] for details.

P . The cases of safe reachability, bounded invariance, and bounded response follow directly from Propositions 6 and 3 and Theorem 3. The remaining cases follow directly from Theorem 2 and Proposition 4. □

7. Closed timed ε-automata

We define the class of *closed timed ε-automata*. These are essentially the timed safety automata of [16] with ε-transitions (silent transitions) [3, 8] and exclusively *closed* invariant and enabling clock constraints.

An example of a closed constraint is $x \leqslant 3$, where x is a clock, as opposed to $x < 3$. Since any timed automaton can be infinitesimally approximated by one with closed constraints [26], this restriction appears to be rather benign in practice, an opinion shared by several researchers (see, e.g., [7]).

As we shall see, this class of timed automata corresponds to finite-state Timed CSP processes. To simplify our exposition, we assume that timed automata cannot communicate ✓.

Let C be a finite set of clocks, denoted x, y, x_1, x_2, etc. The grammar

$$\sigma ::= \mathbf{true} \mid x \leqslant c \mid x \geqslant c \mid x_1 + c_1 \leqslant x_2 + c_2 \mid \sigma_1 \wedge \sigma_2 \mid \sigma_1 \vee \sigma_2$$

defines the set F_C of clock constraints over C. (Here c, c_1, c_2 are non-negative integers.) Note that all constraints are *closed*: interpreted over the non-negative reals, they always define closed subsets in the usual topology.

D 4. *A closed timed ε-automaton is a tuple* $(\Sigma, S, S_0, C, E, \mathsf{inv})$*, where*

- Σ *is a finite alphabet with* $\varepsilon \notin \Sigma$*; we let* $\Sigma^\varepsilon \mathrel{\widehat{=}} \Sigma \cup \{\varepsilon\}$*,*
- S *is a finite set of locations,*
- $S_0 \subseteq S$ *is a set of start locations,*
- C *is a finite set of clocks,*
- $E \subseteq S \times S \times \Sigma^\varepsilon \times \mathcal{P}(C) \times F_C$ *is a finite set of transitions; the components of a transition are, in order: the source location, the target location, the labelling event, the set of clocks to reset, and the enabling clock constraint.*
- $\mathsf{inv} : S \longrightarrow F_C$ *specifies location invariant constraints.*

ε is a special event, invisible to the outside world, which differs from τ in that it is *not* subject to urgency. In other words, the availability of an ε-transition does not block the passage of time.

The class of closed timed ε-automata is denoted **CTA**.

We must now define a suitable semantics for timed automata. Although existing semantics are based on timed traces, we desire a timed refusal trace semantics both to facilitate comparison with Timed CSP and to be able to express basic safety and liveness specifications in a natural and consistent way.

Note that since the semantics of Timed CSP is based on finite downward-closed behaviours, we must give up Alur and Dill's Büchi acceptance conditions [4], and instead consider every location to be accepting.

Except where noted otherwise, the remainder of this discussion considers a fixed closed timed ε-automaton $A = (\Sigma, S, S_0, C, E, \mathsf{inv}) \in$ **CTA**. We re-use and occasionally overload the notation and terminology of Section 4 in a manner which should not cause any problems.

A *clock interpretation* is a function $\nu : C \longrightarrow \mathbb{R}^+$. Clock interpretations allow one to assign truth values to clock constraints in the obvious way; we write $\nu \vDash \sigma$ to indicate that the clock interpretation ν makes the clock constraint σ true. For $\nu : C \longrightarrow \mathbb{R}^+$ a clock interpretation and $t \in \mathbb{R}^+$, we let $\nu + t$ be the clock interpretation such that $(\nu + t)(x) = \nu(x) + t$ for all $x \in C$. For $D \subseteq C$ a set of clocks to be reset, we let $[D := 0]\nu$ be the clock interpretation which evaluates clocks in D to 0 and agrees with ν on clocks outside of D.

A *state* of A is a pair (s, ν), with $s \in S$ a location and ν a clock interpretation.

An operational semantics for closed timed ε-automata can be given using the following two operational rules. Here $t \in \mathbb{R}^+$ and $\gamma \in \Sigma^\varepsilon$.

$$\frac{\forall (0 \leqslant \delta \leqslant t) \,\boldsymbol{.}\, \nu + \delta \models \mathsf{inv}(s)}{(s, \nu) \overset{t}{\rightsquigarrow} (s, \nu + t)} \qquad \frac{(s, s', \gamma, D, \sigma) \in E \quad \nu \models \sigma \wedge \mathsf{inv}(s)}{(s, \nu) \xrightarrow{\gamma} (s', [D := 0]\nu)} \ .$$

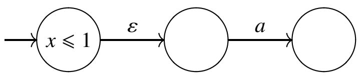
Note that this operational semantics allows transitions into locations where the invariant does not hold; however, when this occurs, no further progress is allowed and a timestop immediately ensues.

Since ε-transitions are not urgent, we consider that their mere availability does not introduce instability, except when the invariant constraint blocks the passage of time; in that case the enabled ε-transitions de facto become urgent (much like τ-transitions in Timed CSP processes) and no stable refusal can be recorded.

Whether or not the limit of the invariant constraint was reached, we also consider a timed automaton to have been unstable at the instant immediately preceding the actual firing of an ε-transition.

To justify these decisions, consider the timed automaton A_1 below.[7]

A_1 :



Assuming that the ε-transition has not yet been taken when time 1 is reached, the invariant constraint blocks further progress of time. Thus the automaton reaches a state from which time cannot pass and no visible transitions can be immediately taken. However, it would clearly be wrong to record a timestop, since the silent ε-transition is forced to occur on the spot, transferring control to a timestop-free location. Note incidentally that recording a timestop at time 1 would also violate clause (3) of the definition of refusal traces (cf. Section 3), to the effect that when time is refused, time stops, at least until a visible event occurs. For this reason, we simply consider that A_1 in its start location is unstable at time 1.

Suppose now that the ε-transition is taken at time 0.5, immediately followed (also at time 0.5) by the a-transition. If we failed to consider that the automaton was unstable at time 0.5 immediately prior to the firing of the ε-transition, we would logically have to conclude that a was stably refusable at time 0.5, yet record its subsequent occurrence immediately afterwards. This would violate clause (2) of the definition of refusal traces, and accordingly justifies our decision to consider any timed automaton to have been in an unstable state immediately prior to the actual firing of an ε-transition.

In line with these modelling assumptions, we postulate that a timed automaton able to perform an infinite sequence of ε-transitions without letting time elapse is in a (potentially) divergent state.

[7] Throughout this paper, we use the following conventions: Start locations are depicted with an incoming arrow not originating from any other location. Enabling clock constraints are decorated with a question mark (?), and invariant constraints are inscribed inside states. The rest of the notation is self-explanatory.

An *execution* of the automaton A is a finite sequence $e = (s_0, \nu_0) \xmapsto{z_1} (s_1, \nu_1) \xmapsto{z_2} \ldots \xmapsto{z_n} (s_n, \nu_n)$, where $s_0 \in S_0$, $\nu_0 = \mathbf{0}$ (the clock interpretation taking each clock to 0), and each subsequence $(s_i, \nu_i) \xmapsto{z_{i+1}} (s_{i+1}, \nu_{i+1})$ of e is either a transition $(s_i, \nu_i) \xrightarrow{\gamma} (s_{i+1}, \nu_{i+1})$ (with $z_{i+1} = \gamma$), or an evolution $(s_i, \nu_i) \overset{t}{\rightsquigarrow} (s_{i+1}, \nu_{i+1})$ (with $z_{i+1} = t$). In addition, every such transition or evolution must be validly allowed by the two operational inference rules listed above. The set of executions of A is written $\mathsf{exec}(A)$.

Given a state (s, ν), let $\mathsf{tref}(s, \nu) \subseteq \mathbb{R}^+ \times (\Sigma^{\checkmark}_{time} \cup \{\bullet\})$ be defined as follows: for any $t \geqslant 0$, if $(s, \nu + t) \not\rightsquigarrow$ and $(s, \nu + t) \xrightarrow{\varepsilon}$, then $(t, \bullet) \in \mathsf{tref}(s, \nu)$. Otherwise, $(t, a) \in \mathsf{tref}(s, \nu)$ if $(s, \nu + t) \overset{a}{\not\longrightarrow}$, and $(t, time) \in \mathsf{tref}(s, \nu)$ if $(s, \nu + t) \not\rightsquigarrow$. (Here and below, $a \in \Sigma$ stands for a visible event.) For $I \subseteq \mathbb{R}^+$ a time interval, let $\mathsf{tref}(s, \nu) \upharpoonright I \mathrel{\widehat{=}} \mathsf{tref}(s, \nu) \cap (I \times (\Sigma^{\checkmark}_{time} \cup \{\bullet\}))$.

We say that (s, ν) *diverges* if (s, ν) has an infinite evolution-less execution whose transitions are exclusively ε's.

Given an execution e of A, we produce an associated canonical refusal trace with divergence $\mathsf{rtd}(e)$ (the largest possible given the execution e), defined inductively on e as follows.

$$\mathsf{rtd}((s, \nu)) \mathrel{\widehat{=}} \begin{cases} (\langle \emptyset \rangle, 0) & \text{if } P \text{ diverges} \\ (\langle \mathsf{tref}(s, \nu) \upharpoonright \{0\} \rangle, \infty) & \text{otherwise} \end{cases}$$

$$\mathsf{rtd}((s, \nu) \xrightarrow{\varepsilon} {}^\frown e) \mathrel{\widehat{=}} \mathsf{rtd}(e)$$

$$\mathsf{rtd}((s, \nu) \xrightarrow{a} {}^\frown e) \mathrel{\widehat{=}} (\langle \mathsf{tref}(s, \nu) \upharpoonright \{0\}, (0, a) \rangle ^\frown T, z) \text{ if } \mathsf{rtd}(e) = (T, z)$$

$$\mathsf{rtd}((s, \nu) \overset{t}{\rightsquigarrow} {}^\frown e) \mathrel{\widehat{=}} (\langle \mathsf{tref}(s, \nu) \upharpoonright [0, t) \rangle \curvearrowright (T + t), z + t) \text{ if } \mathsf{rtd}(e) = (T, z)\ .$$

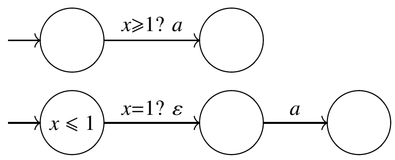
We can thus derive the set of (dense-time) rtd's associated with a timed automaton A: $\mathcal{R}_{\mathbb{R}}[\![A]\!] \mathrel{\widehat{=}} \overline{\downarrow \mathsf{rtd}(\mathsf{exec}(P))}$.

The sets $\mathcal{R}_{\mathbb{Z}}[\![A]\!]$, $\mathcal{T}_{\mathbb{R}}[\![A]\!]$, and $\mathcal{T}_{\mathbb{Z}}[\![A]\!]$, representing respectively the integral-time rtd's of A, the dense-time traces of A, and the integral-time traces of A are all derived from $\mathcal{R}_{\mathbb{R}}[\![A]\!]$ in the same manner as for Timed CSP processes.

Let us now consider some examples. The following two automata enable the event a only after one time unit has elapsed. We give the maximal refusals for a over the interval $[0, 2]$, assuming that a never occurs.

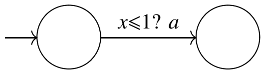
A_2 : $\qquad \langle [0, 1) \times \{a\} \rangle$

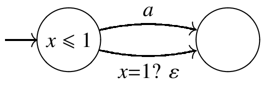
A_3 : $\qquad \langle [0, 1) \times \{a\} \rangle$.



Note that a cannot be refused at time 1. Both A_2 and A_3 are rtd-equivalent to $WAIT\ 1 \mathbin{⨟} a$.

In the next two examples, a is disabled after one time unit has passed. Again assuming that no a is communicated, we give the maximal a-refusals of these timed automata over the interval $[0, 2]$.

A_4 :  $\langle (1,2] \times \{a\} \rangle$

A_5 :  $\langle [1,2] \times \{a\} \rangle$.

Note here that, while A_5 is clearly rtd-equivalent to $a \overset{1}{\rhd} STOP$, no Timed CSP program can possibly be strictly rtd-equivalent to A_4. The reason is that, as discussed in Section 3, maximal refusals of Timed CSP processes always consist of finite unions of refusal tokens (left-closed right-open sets) together with finitely many point refusals. Clearly, no non-trivial left-open set can have this shape. Nevertheless, A_4 and $a \overset{1}{\rhd} STOP$ are rtd-equivalent almost everywhere, as defined below:

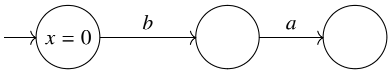
D 5. *Let $A, P \in \mathcal{RTD}$ be sets of rtd's. We say that A and P are* rtd-equivalent almost everywhere *if both the following conditions hold:*

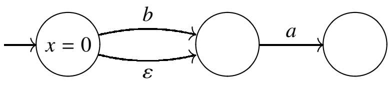
(1) *For every rtd $(T, z) \in A$ there exists an rtd $(T', z) \in P$ with the same divergence value such that T and T' have the same underlying timed trace, and moreover the respective refusals of T and T' differ in at most finitely many points.*
(2) *Vice-versa.*

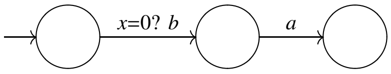
In the case A and P are timed automata and/or Timed CSP programs, this definition applies to $\mathcal{R}_{\mathbb{R}}[\![A]\!]$ and $\mathcal{R}_{\mathbb{R}}[\![P]\!]$.

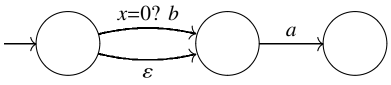
Note that rtd-equivalence almost everywhere implies timed trace equivalence.

In the next four examples, we consider rtd's of timed automata in which the only observed communication is that of b at time 0. Again, we give maximal refusals for a over the interval $[0, 2]$.

A_6 :  $\langle \{(0, a)\}, (0, b), \emptyset \rangle$

A_7 :  $\langle \{(0, \bullet)\}, (0, b), \emptyset \rangle$

A_8 :  $\langle \{(0, a)\}, (0, b), \emptyset \rangle$

A_9 :  $\langle \{(0, a)\}, (0, b), \emptyset \rangle$.

Here A_6 is rtd-equivalent to the signalling process $b \xrightarrow{!} a$ and A_7 is rtd-equivalent to $((b \longrightarrow a) \;\Box\; (c \longrightarrow a)) \setminus c$. In the case of A_7, note that b is necessarily communicated from an unstable state because of the invariant constraint and the availability of an ε-transition. A_8 and A_9, on the other hand, are not unstable prior to communicating b; however, since b is only enabled for an instant, any Timed CSP process mimicking their behaviours will necessarily be unstable.

Thus we see that A_8 is rtd-equivalent almost everywhere to $(b \longrightarrow a) \overset{0}{\triangleright} STOP$, whereas A_9 is rtd-equivalent almost everywhere to $(b \longrightarrow a) \overset{0}{\triangleright} (RANDOM \mathbin{⨟} a)$.

We now give the following standard results and constructions:

P 7. *Closed timed ε-automata are closed under digitization. In other words, for $A \in \mathbf{CTA}$ and $0 \leqslant \delta \leqslant 1$, $[\mathcal{T}_{\mathbb{R}}[\![A]\!]]_\delta \subseteq \mathcal{T}_{\mathbb{R}}[\![A]\!]$.*

A proof can be extracted from the results presented in [15].

Following [2, 4], we now review the definition of *clock regions* and the construction of *region automata*. Let $A = (\Sigma, S, S_0, C, E, \mathsf{inv})$ be a timed automaton. Let k be the largest integer constant appearing in any of the enabling and invariant clock constraints of A. Define an equivalence relation $\approx$ on the set of clock interpretations as follows: $\nu \approx \nu'$ if

(1) For all clocks $x \in C$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$, or both $\nu(x)$ and $\nu'(x)$ are greater than k.

(2) For all $x, y \in C$ with $\nu(x), \nu(y) \leqslant k$, $fract(\nu(x)) \leqslant fract(\nu(y)) \Leftrightarrow fract(\nu'(x)) \leqslant fract(\nu'(y))$.

(3) For all $x \in C$ with $\nu(x) \leqslant k$, $fract(\nu(x)) = 0 \Leftrightarrow fract(\nu'(x)) = 0$.

It is easy to check that $\approx$ partitions the set of clock interpretations into finitely many equivalence classes, termed *clock regions*. Two clock interpretations lie in different equivalence classes if either they differ in the integral parts of the readings of some clock (and one of these numbers is at most k), or if they differ in the ordering of the fractional parts of those clocks having values at most k.

We define a partial order $\preccurlyeq$ on clock regions as follows: $r \preccurlyeq r'$ if, for any $\nu \in r$, there exists $t \in \mathbb{R}^+$ such that $\nu + t \in r'$.

The (untimed) *region ε-automaton $RA(A)$* of A is defined as follows. Its alphabet Σ is the same as that of A; the automaton is also able to perform ε-moves. The states of $RA(A)$ consist of all pairs (s, r), where $s \in S$ is a location of A and r is a clock region of A. The start states of $RA(A)$ consist of all states of the form $(s_0, \{\mathbf{0}\})$, with $s_0 \in S_0$. $RA(A)$ has a transition $(s, r) \xrightarrow{\gamma} (s', r')$ provided that, for every clock interpretation $\nu \in r$ there exist $t \in \mathbb{R}^+$ and some clock interpretation $\nu' \in r'$ such that $(s, \nu) \overset{t}{\rightsquigarrow} (s, \nu + t) \xrightarrow{\gamma} (s', \nu')$ is valid for A.

The relationship between A and $RA(A)$ is given by the following:

P 8. *Every execution of A gives rise to a corresponding unique path in $RA(A)$, and for every path in $RA(A)$ one can find an execution of A which corresponds to it.*

We refer the reader to [4] for the proof.

For r a region, let $\overline{r}$ stand for the closure of r in the Euclidean topology. $\overline{r}$ thus consists of r together with all the regions of lower dimension bounding r. Because closed timed ε-automata have exclusively *closed* invariant and enabling clock constraints, we easily establish the following:

P 9. *Let* $e = (s_0, v_0) \overset{z_1}{\longmapsto} \ldots \overset{z_n}{\longmapsto} (s_n, v_n) \in \mathsf{exec}(A)$ *be an execution of a closed timed* ε*-automaton* A*. Consider a prospective execution* $e' = (s_0, v'_0) \overset{z'_1}{\longmapsto} \ldots \overset{z'_n}{\longmapsto} (s_n, v'_n)$ *which is identical to* e *except possibly for the durations of the various evolutions in* e*. Assume that the clock interpretations of* e' *are consistent with its evolutions, and that, for all* $0 \leqslant i \leqslant n$*, whenever* $v_i \in r$ *then* $v'_i \in \overline{r}$*. Then* $e' \in \mathsf{exec}(A)$*.*

Lastly, we record the following observation:

P 10. *Let* $A \in \mathbf{CTA}$*. Then for any states* $(s, v), (s, v')$ *of* A *with* $v \approx v'$*,* $(s, v) \rightsquigarrow$ *if and only if* $(s, v') \rightsquigarrow$*, and for any* $\gamma \in \Sigma^{\varepsilon}$*,* $(s, v) \overset{\gamma}{\longrightarrow}$ *if and only if* $(s, v') \overset{\gamma}{\longrightarrow}$*.*

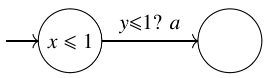
8. Timed automata as Timed CSP processes

Given any closed timed ε-automaton $A \in \mathbf{CTA}$, we construct two corresponding finite-state Timed CSP processes $P^A_{\mathbb{R}}$ and $P^A_{\mathbb{Z}}$, capturing respectively the dense-time and integral-time behaviours of A. Our constructions use some ideas introduced in [10].

We begin by giving the construction of $P^A_{\mathbb{R}}$. Let $A = (\Sigma, S, S_0, C, E, \mathsf{inv})$ be an automaton with m clocks $x_1, x_2, \ldots, x_m \in C$. We build $P^A_{\mathbb{R}}$ as the parallel composition of $m+2$ processes over alphabet $\Sigma \cup \{\checkmark, \varepsilon\}$: a network *CLOCKS* of m processes for the clocks, a process *REGIONS* to mimic the clock regions graph, and a process *LOCATIONS* which switches between discrete states corresponding to locations of A.

Taken together, *LOCATIONS* and *REGIONS* simulate the region automaton $RA(A)$, whereas the *CLOCKS* network provides exact timing information, ensuring that the process remains in each region exactly as long as it is supposed to.

One might think that the *REGIONS* process is superfluous, given the network *CLOCKS* of accurate clocks. Unfortunately, the phenomenon of point nondeterminism introduces certain difficulties. Consider, for instance, the timed automaton A_{10} equipped with two clocks x and y:

A_{10} : 

.

A_{10} cannot exhibit a timestop: if the event a has not occurred within one time unit, then the invariant constraint forces a to happen at time 1. Since x and y both start with value 0, they both reach time 1 together, ensuring that the a-transition is never disabled.

One might attempt to represent the timed automaton A_{10} with the help of two clocked processes, one representing the invariant constraint, which does nothing for one time unit and then blocks the passage of time until a visible action occurs, and another which offers the event a for one time unit, then withdraws the offer: $P = (STOP \overset{1}{\rhd} TIMESTOP) \,\Box\, (a \overset{1}{\rhd} STOP)$.

Unfortunately, P can exhibit behaviours that are impossible for A_{10}: at time 1, P's right-hand component may choose to make a τ-transition into *STOP*, which then inevitably creates a timestop. Thus even though the left-hand and right-hand components are essentially deterministic[8] processes running at exactly the same rate, their combination exhibits nondeterminism as a result of the infinitesimal (in fact, instantaneous) uncertainty of the clocks at time 1. If, however, we use an underlying regions construction which, through hidden synchronizations, effectively straightjackets both clocks into a single one, then the resulting process (equivalent to $a \overset{1}{\rhd} a \overset{!}{\longrightarrow} STOP$) precisely corresponds to the timed automaton A_{10}.

We model each clock region r as a process *REGr*. *REGr* is at any time willing to accept, on some internal (i.e., globally hidden) channel, the event *query.r* from *LOCATIONS*; upon entering a new location, *LOCATIONS* can thus always check whether or not the invariant constraint is satisfied. Now suppose that r' is the region immediately following r temporally. The region-process *REGr* is at any time willing to accept, again on some internal channel, the command (i.e., event) *switch*.r', which transfers control to the region-process *REGr*$'$, as well as the event *preswitch*.r', which announces that such a move is imminent. The events *switch*.r' and *preswitch*.r' are initiated by *CLOCKS*, and also require the synchronized participation of *LOCATIONS*, as a means to enforce the location invariant constraints. A region-process is also at any time willing to accept any one of the commands *reset*.x_i, again on some internal channel, and subsequently transfer control to the process associated with the region reached by resetting x_i. Lastly, for any $\gamma \in \Sigma^{\varepsilon}$, the region-process *REGr* is always willing to accept the 'r-tagged event' $r.\gamma$. Although this latter communication is external (not hidden) for $\gamma \neq \varepsilon$, a subsequent global renaming operation restores all such communications to their nominal values (the simple event γ in this case). The composite clock regions process is denoted *REGIONS*.

Recall the constant k, which is the largest integer appearing in any of the enabling and invariant clock constraints of A. We model each clock x_i as a process CLx_i. This process can be in any of the $2k+2$ following discrete states (represented as disjoint subsets partitioning $\mathbb{R}^+$): $\{0\}, (0, 1), \{1\}, (1, 2), \ldots, (k-1, k), \{k\}, (k, \infty)$. CLx_i switches from discrete state to discrete state by means of the interrupt operator; it spends unit-duration periods in bounded 'interval' states, and zero-duration periods in 'singleton' states. While in a given discrete state, CLx_i is always willing to synchronize on events of the form *preswitch.r* and *switch.r*, where r is a region compatible with the current discrete state of CLx_i. Immediately prior to entering a new discrete state, CLx_i signals a choice of all events of the form *switch*.r', where r' is any region compatible with the new discrete state. We stress that one of these events (or a *reset*, see below) must eventually be accepted, otherwise a timestop ensues. (Events of the form *switch.r*, where r is a region compatible with the discrete state of CLx_i, are still allowed, but do not disable the urgent offering of the *switch*.r' events.) Note that since all clocks, as well as the *REGr* process,

[8] The study of determinism and nondeterminism in Timed CSP can be an intricate and subtle affair, with the present situation showing only the tip of the iceberg; we refer the reader to [30] for more details on the matter.

must synchronize on *switch* events, in particular all the clocks for which entering region r' would correspond to a change of discrete state must agree to the move $switch.r'$; this is the 'straightjacket' mechanism which keeps all the clocks in the right region at all times. While in this urgent configuration, CLx_i is also willing to communicate matching $preswitch.r'$ events any number of times. These events need not occur, but if one does, all the clocks for which entering region r' would correspond to a change of discrete state must again agree to it. Lastly, CLx_i is at any time—even while offering a *switch*—willing to accept the command $reset.x_i$, which prompts the jump to state $\{0\}$. The parallel composition of all the CLx_i's is denoted *CLOCKS*.

Any clock constraint $\sigma \in F_C$ can be identified with a subset $\{r \mid r \models \sigma\}$ of clock regions. Whenever the timed automaton A offers a visible event a under a particular clock interpretation ν, the process $P^A_{\mathbb{R}}$ is meant to offer a corresponding region-tagged event $r.a$, where r is the region corresponding to ν. Recall that this transition is subsequently renamed to a at the outermost level. As we shall see, ε-transitions are handled slightly differently.

The process *LOCATIONS* mimics the transfer of control within the various locations of A. Initially, *LOCATIONS* nondeterministically begins in one of the start locations in S_0. Upon entering a new location $s \in S$, the first thing *LOCATIONS* does is signal to *REGIONS* a choice of events of the form $query.r$, where $r \in \mathsf{inv}(s)$. If *REGIONS* cannot synchronize on any of these events, then a location has been reached for which the invariant constraint is violated, and a timestop automatically ensues. Otherwise, while in location s, *LOCATIONS* is always willing to accept events of the form $switch.r$, as long as $r \in \mathsf{inv}(s)$. For every transition $e = (s, s', a, D, \sigma)$ with $a \in \Sigma$ a visible event, *LOCATIONS*, while in location s, continuously offers a choice of region-tagged events of the form $r.a$, where $r \in \sigma$. If any of these transitions is accepted, then *LOCATIONS* immediately proceeds to signal the events $reset.x$, for every clock $x \in D$ to be reset. Finally, it enters the new location s', and the cycle begins anew.

Note that *LOCATIONS* cannot handle ε-transitions in the same way as visible events. Indeed, since ε's are meant to be globally hidden, anytime an ε-transition is offered it blocks the passage of time. The solution is therefore to delay offers of ε-transitions by some nondeterministically chosen amount of time, which is achieved by writing $\ldots \;\Box\; (RANDOM \;\text{⨟}\; r.\varepsilon \longrightarrow \ldots) \;\Box\; \ldots$ in place of $\ldots \;\Box\; (r.\varepsilon \longrightarrow \ldots) \;\Box\; \ldots$. However, we still have to ensure the availability of valid ε-transitions whenever the alternative is a timestop (brought on by the imminent expiration of the current invariant constraint). To this end, whenever *LOCATIONS* reaches the limit of $\mathsf{inv}(s)$, we interrupt the *RANDOM* delaying and offer the ε-transition immediately. This can be achieved as follows:

$$\begin{aligned} &\ldots \;\Box \\ &\Big((RANDOM \;\text{⨟}\; r.\varepsilon \longrightarrow \ldots) \;\Box\; \Box\{preswitch.r' \longrightarrow r.\varepsilon \longrightarrow \ldots \mid r' \notin \mathsf{inv}(s)\}\Big) \\ &\hfill \Box \ldots \;. \end{aligned}$$

LOCATIONS, *REGIONS*, and *CLOCKS* are combined in parallel and required to synchronize on all appropriate events. Events on internal channels and ε-events are then hidden. Finally, a global renaming operator converts all communications of the form $r.a$ back to their nominal Σ-value. The resulting process is denoted $P^A_{\mathbb{R}}$. It should be clear from our description that $P^A_{\mathbb{R}}$ is finite-state (Definition 3).

We note that our modelling assumptions for Timed CSP entail an infinitesimal mismatch between the clock valuations corresponding to the continuous states of *CLOCKS* and the clock regions of *REGIONS*. For example, a given region r may correspond to clock x being strictly greater than 1, whereas *REGIONS* will be able to enter the state-region r when $x = 1$. (We observed a similar phenomenon when analyzing the timed automaton A_4 in the previous section.) In general, for any region r, if the *REGIONS* process is in state-region r, then the actual clock interpretation corresponding to the current continuous state of *CLOCKS* lies somewhere in $\overline{r}$.

Thanks to Propositions 8 and 9, it follows that the timed automaton A and the Timed CSP program $P^A_{\mathbb{R}}$ have exactly the same timed traces. Note that this provides an alternate proof of Proposition 7 (to the effect that closed timed ε-automata are closed under digitization), in view of Proposition 3.

Since ε-transitions of A become τ-transitions in $P^A_{\mathbb{R}}$, whenever A potentially diverges then so does $P^A_{\mathbb{R}}$. On the other hand, if A is livelock-free then it is not difficult to see that all of $P^A_{\mathbb{R}}$'s internal chatter is of bounded length. Thus A and $P^A_{\mathbb{R}}$ have exactly the same livelock behaviours.

Finally, if A is livelock-free, then our construction together with Propositions 2 and 10 imply that A and $P^A_{\mathbb{R}}$ have exactly the same refusals, except for punctual instances of: (i) infinitesimal regions mismatch, as described earlier, and (ii) instability in $P^A_{\mathbb{R}}$ due to the presence of τ-transitions. Since there can only be a finite number of such occurrences in any bounded non-livelocking computation, we have the following result:

T 5. *Let $A \in$ **CTA** be a closed timed ε-automaton. Then $P^A_{\mathbb{R}} \in$ **TCSP** is a finite-state Timed CSP program which is rtd-equivalent almost everywhere to A.*

In particular, $P^A_{\mathbb{R}}$ and A have the same timed traces: $\mathcal{T}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!] = \mathcal{T}_{\mathbb{R}}[\![A]\!]$.

We immediately deduce the following:

C 1. *Let $A \in$ **CTA**, and let S be any safety specification (set of allowable timed traces). Then $A \vDash^{\mathcal{T}}_{\mathbb{R}} S$ if and only if $P^A_{\mathbb{R}} \vDash^{\mathcal{T}}_{\mathbb{R}} S$.*

The relationship between A and $P^A_{\mathbb{R}}$ is arguably even stronger than Theorem 5 implies, as the following result suggests:

T 6. *Let $A \in$ **CTA**, and let $S \subseteq \mathcal{RTD}$ be either of the two liveness specifications considered in Section 6: timestop-freedom (TSF) or constant availability of the event a ($AVAIL_{\{a\}}$). Then the following are equivalent:*

(1) A satisfies S over dense time: $A \vDash^{\mathcal{R}}_{\mathbb{R}} S$.

(2) $P^A_{\mathbb{R}}$ satisfies S over dense time: $P^A_{\mathbb{R}} \vDash^{\mathcal{R}}_{\mathbb{R}} S$.

(3) $P_{\mathbb{R}}^{A}$ *satisfies* S *over discrete time:* $P_{\mathbb{R}}^{A} \vDash_{\mathbb{Z}}^{\mathcal{R}} S$.

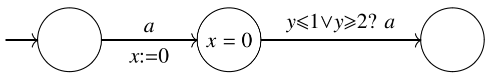
We recall that the last of these is a discrete check which can be performed (under certain conditions) on the model checker FDR.

P . The equivalence of (2) and (3) was established in Theorem 4. (1) implies (2) since the immediate refusals of any stable configuration (s, r, v) of $P_{\mathbb{R}}^{A}$ (with s a location of A, r a clock region, and v a clock interpretation) are entirely determined by the pair (s, r) alone. It thus remains to show that (2) implies (1).

We tackle the case $S = AVAIL_{\{a\}}$, timestop-freedom being handled in an entirely similar way. Suppose that $P_{\mathbb{R}}^{A}$ never records a refusal of a, yet that on some execution, A reaches a state (s, v) which immediately and stably refuses a. Stability implies that either (i) $(s, v) \overset{\varepsilon}{\not\longrightarrow}$, or (ii) v has not yet reached the limit of $\mathsf{inv}(s)$. Let r be the region containing v. $P_{\mathbb{R}}^{A}$ can clearly reach the configuration (s, r, v), from which a is not possible. Since $P_{\mathbb{R}}^{A}$ nonetheless fails to record a refusal of a, $P_{\mathbb{R}}^{A}$ in configuration (s, r, v) must always have some τ-transitions pending. We may assume that none of these τ-transitions originate from an ε-transition: indeed, either (i) there are none available, or (ii) since v has not yet reached the limit of $\mathsf{inv}(s)$, we can invoke our nondeterministic delaying construction to conclude that ε-transitions may not be immediately available. All enabled τ-transitions therefore correspond to chatter on $P_{\mathbb{R}}^{A}$'s internal channels, and must eventually lead to a stable configuration (s, r', v). Note that the location s and clock valuation v cannot have changed since no visible or ε-transitions were taken. The stability of configuration (s, r', v) entails that the pair (s, r') cannot refuse a (otherwise $P_{\mathbb{R}}^{A}$ could record this refusal). Since enabling constraints are closed, we deduce that a must be enabled (in location s) over the whole of $\overline{r'}$. On the other hand, we clearly have $v \in \overline{r'}$, and therefore the automaton A cannot refuse a in state (s, v), contradicting our earlier assumption. $\square$

It is interesting to note that in the case of closed timed ε-automata, liveness specifications such as the ones considered above *cannot* be established over dense time simply by checking whether they hold over discrete time. To see this, consider the following timed automaton, equipped with two clocks x and y:

A_{11} :



.

It is plain that A_{11} will timestop if the first transition is taken at any time strictly between 1 and 2. Note, however, that the *integral* rtd's $\mathcal{R}_{\mathbb{Z}}[\![A_{11}]\!]$ of A_{11} do not exhibit any timestop.

Having noticed such phenomena, Bošnački comments in [9] that digitization techniques appear to be inadequate to handle the requirement of timestop-freedom. Theorem 6 shows that this is not the case; and indeed, $\mathcal{R}_{\mathbb{Z}}[\![P_{\mathbb{R}}^{A_{11}}]\!]$ contains the refusal trace $\langle \emptyset, (1, a), \{1\} \times \Sigma_{time}^{\checkmark} \rangle$. (Again, the reason we do have this refusal trace is the instantaneous uncertainty due to point nondeterminism.)

Note, however, that when it comes to timed automata, our method is certainly no more efficient (and undoubtedly less general) than other algorithms directly based on the region automaton construction (see, e.g., [16]). Rather, the main point we are making is that directly using Timed CSP to model and verify systems offers many advantages, not least of which is the applicability of digitization techniques for both safety and liveness properties, as discussed in Section 6.

Nonetheless, for the purposes of modelling closed timed ε-automata using Timed CSP, the region automaton-based construction we have given should probably be optimized. In particular, in most cases (depending on the automaton and the specification to be verified), many regions can be safely discarded. Since such investigations have in the past already received a significant amount of attention (see, e.g., [14]), we focus on the special case of safety properties closed under inverse digitization.

The construction of $P^A_{\mathbb{Z}}$, meant to capture integral-time traces of A, is very similar to that of $P^A_{\mathbb{R}}$. The essential difference is that the *REGIONS* process is much coarser (and correspondingly so are the other two components): the only clock regions considered are those of the form $\{(j_1, j_2, \ldots, j_m)\}$, i.e., integral singletons in $(\mathbb{R}^+)^m$. This saves us a factor of approximately $m! \cdot 4^m$ in the number of regions over our previous construction. The basic mechanisms to ensure the proper running of the process and to enforce the satisfaction of the invariant and enabling constraints (on integral behaviours) are engineered in the obvious way along the lines of those of $P^A_{\mathbb{R}}$. The resulting process is denoted $P^A_{\mathbb{Z}}$.

T 7. *For any timed automaton $A \in$ **CTA**, $P^A_{\mathbb{Z}}$ and A have the same integral-time timed traces: $\mathcal{T}_{\mathbb{Z}}[\![P^A_{\mathbb{Z}}]\!] = \mathcal{T}_{\mathbb{Z}}[\![A]\!]$.*

The significance of this result comes from the applicability of digitization techniques to the timed trace verification problem, as detailed in Section 6.

9. Timed CSP processes as timed automata

A legitimate question is whether Timed CSP is any more expressive than closed timed automata. Since Timed CSP is Turing-complete (it can encode infinite counters), the answer must be affirmative. However, when restricted to finite-state processes (as defined in Section 5), we find that Timed CSP is exactly as expressive as closed timed ε-automata.

T 8. *Let $P \in$ **TCSP** be a finite-state program. Then there exists a closed timed ε-automaton $A \in$ **CTA** which is rtd-equivalent to P and $P^A_{\mathbb{R}}$: $\mathcal{R}_{\mathbb{R}}[\![A]\!] = \mathcal{R}_{\mathbb{R}}[\![P]\!] = \mathcal{R}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!]$.*

Note that we have *total* rtd-equivalence here, as opposed to rtd-equivalence almost everywhere.

P . Let P be a finite-state program over some alphabet Σ, and substitute every occurrence of *RANDOM* in P by the term $\varepsilon \longrightarrow SKIP$, where $\varepsilon \notin \Sigma$ is a newly introduced event. The resulting program, which we call P^ε, is clearly also finite-state.

Consider the set of terms arising from arbitrary executions of P^ε, and let S be the finite collection of equivalence classes of this set modulo $\sim$. (Recall that $\sim$ identifies Timed CSP terms which may differ in the various values of their respective delays, but are otherwise syntactically identical.) Let S_0 be the singleton containing the equivalence class of P^ε.

The operational rules for Timed CSP enable us to view S as the set of nodes of a finite labelled transition system, with a transition $\xrightarrow{\gamma}$ between two equivalence classes if there exist two representatives from these equivalence classes that have a $\xrightarrow{\gamma}$ transition between them. We now construct a set of clocks C as follows. To begin with, we postulate a clock $x_0 \in C$. Next, consider the set of timeout ($\overset{t}{\rhd}$) and interrupt ($\overset{t}{\lightning}$) operators occurring in P^ε (viewed as a representative of its equivalence class), and index these operators with successive positive integers, starting at 1. For each of these operators, add an associated clock x_i to C. Continue exploring the labelled transition system S in breadth-first search fashion. Note that the only transitions able to introduce new temporal operators are τ-transitions corresponding to recursion unwinding. Associate to every such freshly introduced temporal operator a fresh index, and add a corresponding clock to C. On the other hand, temporal operators carried over from previous equivalence classes keep their original indices. (Note that when generating the labelled transition system, the operational rules allow us to identify unambiguously the origin of every operator.)

For every indexed temporal operator $\overset{t}{\rhd}_i$ or $\overset{t}{\lightning}_i$, let $c_i \in \mathbb{N}$ be the highest value assumed by t.

We say that an indexed temporal operator $\overset{t}{\rhd}_i$ or $\overset{t}{\lightning}_i$ is *active* in a given equivalence class if either its delay argument t assumes more than one value over the terms of that class, or (in case $t = 0$) the terms of that class allow the corresponding τ-transition to fire immediately. For example, $\overset{t}{\rhd}_i$ is active in $(a \overset{t}{\rhd}_i b) \Box c$ but not in $a \longrightarrow ((a \overset{t}{\rhd}_i b) \Box c)$. (Activity can also be defined directly by structural induction on the syntax of terms.)

The required automaton corresponding to P is $A = (S, S_0, C, E, \mathsf{inv})$, where we now specify the transitions of A together with enabling and invariant constraints. For every non-τ transition (including ε-transitions) between equivalence classes in S, add a like-labelled transition in A with the same source and target, and with enabling constraint **true**. For every τ-transition created by some i-indexed temporal operator, add an ε-labelled transition in A with the same source and target, and with enabling condition $x_i = c_i$. For every other τ-transition, add an ε-labelled transition in A with the same source and target, and with enabling condition $x_0 = 0$.

For every i-indexed active temporal operator in a given equivalence class, conjoin the constraint $x_i \leqslant c_i$ to the invariant of the corresponding location of A. Moreover, if the equivalence class has any τ-transitions which do not originate from a temporal operator, conjoin also the constraint $x_0 = 0$ to the invariant.

Every transition of A resets the clock x_0. Moreover, when taking a transition into an equivalence class in which some i-labelled temporal operator becomes freshly active, reset clock x_i on the corresponding transition of A.

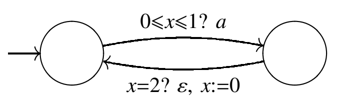
The equivalence $\mathcal{R}_{\mathbb{R}}[\![A]\!] = \mathcal{R}_{\mathbb{R}}[\![P]\!] = \mathcal{R}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!]$ now follows by construction. The reason we have total rtd-equivalence, rather than mere rtd-equivalence almost everywhere, is that every unstable behaviour of P yields an unstable behaviour of A. The mismatches highlighted by automata A_4, A_8, and A_9 of Section 7 do not arise because once enabled, visible events in A can only be disabled by an ε-transition, creating instability. □

We note that a similar result was proved by Jackson in his doctoral dissertation [18]. More precisely, Jackson first severely restricted the syntax of Timed CSP to ensure that all allowable programs were de facto finite-state. He then translated such programs into *action-timed graphs* [21], which are essentially timed ε-automata. Although his very detailed and careful construction differs significantly from ours, the end result is rather similar. Theorem 8 improves on Jackson's work mainly in that we impose *semantic*, rather than *syntactic*, restrictions on Timed CSP: the programs we consider are all required to be finite-state. It is clear that this restriction is both necessary and sufficient.

It is interesting to ask whether the results presented in this section and the previous one can be tightened further in any way. We note that since arbitrary timed automata are in general not closed under digitization, the restriction to closed timed automata is necessary.[9]

We have observed a very slight discrepancy (rtd-equivalence *almost everywhere*) in general between the rtd's of a closed timed ε-automaton A and its corresponding Timed CSP program $P^A_{\mathbb{R}}$. While Corollary 1 and especially Theorem 6 suggest that this mismatch is of little consequence, it is worth investigating ways in which it could be eliminated. In view of Theorem 8, one obvious solution would be to restrict ourselves to closed timed ε-automata in which enabling constraints for visible transitions are never bounded above. Another solution would be to artificially alter the denotational semantics of closed timed ε-automata so as to guarantee conformance with Timed CSP, through the introduction of instability in various places. All things considered, both alternatives seem somewhat contrived, and are less satisfactory than the results of the current paper.

Another question is whether we can dispense with silent transitions. To answer this, consider the following ε-timed automaton A:

A : 

.

A can perform an arbitrary number of a's, subject only to the restriction that the ith a should appear in the time interval $[2i-2, 2i-1]$. Using the results of [17], it is possible to show that A is not equivalent to any timed automaton deprived

[9] In fact, it turns out that it is decidable whether an arbitrary timed automaton is closed under digitization or not, and when it is, the timed automaton is always equivalent to a closed timed automaton [26].

of silent transitions. Intuitively, this is because in order to keep track of all time intervals of the form $[2i-2, 2i-1]$, either an infinite number of clocks must be used, or some clock or clocks must periodically be reset at integral points in time. Of course, the timed traces of A are easily captured with a Timed CSP program such as

$$P = \left(\left(b \longrightarrow ((a \longrightarrow P) \overset{1}{\rhd} \mathit{STOP})\right) \underset{\{b\}}{\|} (\mu X \,.\, b \longrightarrow \mathit{WAIT}\ 2 \,;\, X)\right) \setminus b \ .$$

In view of this example, we could weaken our question and ask whether *urgent* silent transitions are sufficient for timed automata to achieve the expressive power of Timed CSP—after all, Timed CSP's silent τ-transitions are themselves all urgent. However, consider the process $\mathit{RANDOM} \,;\, a$, which has, among others, the refusal trace $\langle [0, 0.5) \times \{a\}, (0.5, a), \emptyset \rangle$. It is not difficult to see that no τ-automaton (i.e., timed ε-automaton for which ε-transitions must occur as soon as they are enabled) can possibly exhibit such a refusal trace. Indeed, since all enabling constraints are required to be integral, for the first event communicated to occur at time 0.5 it must have been enabled since time 0, ruling out the initial refusal set.

Interestingly, the constructions given in Theorems 5 and 8 reveal that the respective expressive powers of Timed CSP without *RANDOM* and closed timed τ-automata are rtd-equivalent almost everywhere.

We remark that the process *RANDOM* was added to Timed CSP's syntax as an *underspecification* mechanism; it allowed us, among other things, to capture key specification processes such as *TSF*, representing timestop-freedom. However, much like non-urgent ε-transitions, it is unlikely that *RANDOM* would figure in any real implementation.

Although we have shown that both *RANDOM* and ε-transitions are necessary to capture certain specific refusal behaviours, it is a very interesting open question whether they increase timed trace expressiveness at all.

C 1. *The timed trace expressiveness of Timed CSP and Timed CSP without RANDOM are identical, and likewise for timed ε-automata and timed τ-automata:*

*(1) For any $P \in$ **TCSP**, there exists $P' \in$ **TCSP** not containing the subterm RANDOM such that $\mathcal{T}_{\mathbb{R}}[\![P]\!] = \mathcal{T}_{\mathbb{R}}[\![P']\!]$.*

(2) For any timed ε-automaton A there exists some timed τ-automaton A' such that $\mathcal{T}_{\mathbb{R}}[\![A]\!] = \mathcal{T}_{\mathbb{R}}[\![A']\!]$.

As discussed above, (2) implies (1), but not vice-versa, since we are not restricting ourselves to *closed* timed automata in this conjecture.

10. Conclusion and future work

We have characterized the expressive power of finite-state Timed CSP processes as that of closed timed automata—timed safety ε-automata [3, 16, 8] with closed invariant and enabling clock constraints.

We have also shown that Timed CSP, as augmented in this paper, is expressive enough to capture some of the most important specifications on timed systems as refinements. Such specifications include safe reachability, bounded invariance, and bounded response, as well as the liveness properties of timestop-freedom and constant availability.

We have also established some important properties enjoyed by Timed CSP, which have enabled us to show that all the specifications listed above can be verified through digitization analysis. To this end, it is possible, under certain conditions, to use the model checker FDR, as detailed in [23, 24].

A number of questions remain open. One concerns the expressiveness of Timed CSP as a *specification* formalism. Following the lead of [19] in the untimed case, it would be interesting to determine precisely which fragment of a quantitative linear-time temporal logic such as MTL can be captured through refinement. It would also be interesting to systematically identify which liveness properties, such as timestop-freedom and constant availability, are guaranteed to hold over dense time whenever they hold over discrete time. Lastly, we would like to settle Conjecture 1, to the effect that the addition of the *RANDOM* process, or of non-urgent ε-transition in the case of timed automata, do not increase timed trace expressiveness.

As well as pursuing the above research topics, we are actively engaged in applying our results to case studies.

Acknowledgements

The first author was supported by the Defense Advanced Research Project Agency (DARPA) and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485, and the Office of Naval Research (ONR) under contract no. N00014-95-1-0520. The second author was supported by ONR and NSF. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, ARO, ONR, NSF, the U.S. Government or any other entity.

References

[1] A , B. S , F. B. 1985. Defining Liveness. *Information Processing Letters 21*, 4, 181–185.

[2] A , R., C , C., D , D. 1993. Model-Checking in Dense Real-Time. *Information and Computation 104*, 1, 2–34.

[3] A , R. D , D. 1990. Automata for Modeling Real-Time Systems. In *Proceedings of ICALP'90*, Volume 443 of *LNCS*. Springer, 322–335.

[4] A , R. D , D. 1994. A Theory of Timed Automata. *Theoretical Computer Science 126*, 2, 183–235.

[5] A , R., F , L., H , T. A. 1999. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science 211*, 253–273.

[6] A , R. H , T. A. 1993 Real-Time Logics: Complexity and Expressiveness. *Information and Computation 104*, 1, 35–77.

[7] A , E., M , O., P , A. 1998. On Discretization of Delays in Timed Automata and Digital Circuits. In *Proceedings of CONCUR'98*, Volume 1466 of *LNCS*. Springer, 470–484.

[8] B´ , B., D , V., G , P., P , A. 1998. Characterization of the Expressive Power of Silent Transitions in Timed Automata. *Fundamenta Informaticae 36*, 2-3, 145–182.
[9] B ˇ ˇ . D. 1999. Digitization of Timed Automata. In *Proceedings of FMICS'99*, 283–302.
[10] C , R. G , M. 1995. Translating Timer Automata to TCSP. Formal Systems Design and Development, Inc.
[11] C , E. M., G , O., P , D. A. 1999. *Model Checking.* MIT Press, Cambridge, MA.
[12] D , J. 1991. *Specification and Proof in Real-Time Systems.* PhD thesis, Oxford University.
[13] D , J. S , S. A. 1990. Factorising Proofs in Timed CSP. In *Proceedings of MFPS'90*, Volume 442 of *LNCS*. Springer, 129–159.
[14] H , T. A., K , O., V , M. Y. 1996. A Space-Efficient On-The-Fly Algorithm for Real-Time Model Checking. In *Proceedings of CONCUR'96*, Volume 1119 of *LNCS*. Springer, 514–529.
[15] H , T. A., M , Z., P , A. 1992. What Good Are Digital Clocks? In *Proceedings of ICALP'92*, Volume 623 of *LNCS*. Springer, 545–558.
[16] H , T. A., N , X., S , J., Y , S. 1994. Symbolic Model Checking for Real-Time Systems. *Information and Computation 111*, 2, 193–244.
[17] H , P. 1998. Timed Automata and Recognizability. *Information Processing Letters 65*, 6, 313–318.
[18] J , D. M. 1992. *Logical Verification of Reactive Software Systems.* PhD thesis, Oxford University.
[19] L , M., M , T., C , A. 2001. How to Make FDR Spin: LTL Model Checking Using Refinement. In *Proceedings of FME'01*, Volume 2021 of *LNCS*. Springer, 99–118.
[20] M , A. 1992. *A Refusal Testing Model for CSP.* PhD thesis, Oxford University.
[21] N , X., S J., Y , S. 1991. From ATP to Timed Graphs and Hybrid Systems. In *Proceedings of REX Workshop 91*, Volume 600 of *LNCS*. Springer, 549–572.
[22] O , J. 2001. Axiomatizing Timed CSP. In preparation.
[23] O , J. 2001. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems.* PhD thesis, Oxford University. Technical report PRG-RR-01-06.
[24] O , J. 2002. Digitisation and Full Abstraction for Dense-Time Model Checking. In *Proceedings of TACAS'02*, Volume 2280 of *LNCS*. Springer, 37–51.
[25] O , J. W , J. B. 2002. Towards Specification as Refinement in Timed Systems. In *Proceedings of AVoCS'02.* Technical report CSR-02-6, School of Computer Science, the University of Birmingham.
[26] O , J. W , J. B. 2003. Revisiting Digitization, Robustness, and Decidability for Timed Automata. In *Proceedings of LICS'03.* IEEE Computer Society Press.
[27] P , I. 1987. Refusal Testing. *Theoretical Computer Science 50*, 3, 241–284.
[28] R , G. M. 1988. *A Mathematical Theory for Real-Time Distributed Computing.* PhD thesis, Oxford University.
[29] R , G. M. R , A. W. 1986. A Timed Model for Communicating Sequential Processes. In *Proceedings of ICALP'86*, Volume 226 of *LNCS*. Springer, 314–323.
[30] R , G. M. R , A. W. 1999. The Timed Failures-Stability Model for CSP. *Theoretical Computer Science 211*, 85–127.
[31] R , A. W. 1997. *The Theory and Practice of Concurrency.* Prentice-Hall International, London.
[32] S , S. A. 1989. *Correctness and Communication in Real-Time Systems.* PhD thesis, Oxford University.
[33] S , S. A. 1994. Using CSP with Z in the Mine Pump Case Study. Unpublished.
[34] S , S. A. 1995. An Operational Semantics for Timed CSP. *Information and Computation 116*, 2 (Feb.), 193–213.
[35] S , S. A. 2000. *Concurrent and Real Time Systems: The CSP Approach.* John Wiley.
[36] Y , Y., M , P., L , L. 1999. Model Checking TLA$^+$ Specifications. In *Proceedings of CHARME'99*, Volume 1703 of *LNCS*. Springer, 54–66.