URL: http://www.elsevier.nl/locate/entcs/volume68.html 18 pages

Timed CSP = Closed Timed Automata[1]

Joël Ouaknine[2,3] and James Worrell

*Department of Mathematics, Tulane University,
New Orleans LA 70118, USA*

Abstract

We study the expressive power of an augmented version of Timed CSP and show that it is precisely equal to that of closed timed automata—timed automata with closed invariant and enabling clock constraints. We also show that this new version of Timed CSP is expressive enough to capture the most widely used specifications on timed systems as refinements between processes, and moreover that refinement checking is amenable to digitisation analysis. As a result, we are able to verify some of the most important timed specifications, including branching-time liveness properties such as timestop-freedom and constant availability, using the model checker FDR (a commercial product of Formal Systems (Europe) Ltd.).

1 Introduction

The formal analysis of real-time systems usually involves both an implementation 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 [4] as implementation formalism, and some (quantitative) temporal logic to express specifications [6]. Verification can then be carried out using model checking [3]. In Reed and Roscoe's dense-time process algebra Timed CSP [22,21], specifications usually consist of allowable sets of behaviours, often described mathematically. Verifying that processes meet their specifications then usually proceeds through some kind of proof system [24,11,25].

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). The only instance we are aware of is the use of event-clock automata to express specifications of (arbitrary) timed automata [5]. More recently, we have shown

[1] This research was supported by the U.S. ONR and the NSF.

[2] Now at CMU School of Computer Science, 5000 Forbes Ave., Pittsburgh PA 15213, USA.

[3] Email: joelo@andrew.cmu.edu

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 [16,17]. The main disadvantage of a refinement-based approach for Timed CSP is that expressiveness 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 [11]; 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.

As a result, we are able to show that the most widely used specifications on timed systems can be captured as refinements between Timed CSP processes, and moreover that such refinements can be verified on the model checker FDR by means of digitisation techniques. In fact, we even show that a number of branching-time liveness properties such as timestop-freedom and constant availability can be verified through digitisation (and FDR), in contrast to the situation with timed automata.

We characterise the expressive power of this augmented version of Timed CSP (when restricted to finite-state processes) as precisely that of closed timed automata. Closed timed automata are the timed safety automata of [13] with exclusively closed invariant and enabling clock constraints (of the form $x \leqslant 3$ rather than $x < 3$, for example). Timed CSP appears to be the most general modelling formalism systematically yielding processes closed under digitisation (and thus amenable to digitisation techniques), making it a prime candidate for the practical formal analysis of timed systems.

2 Timed CSP Syntax and Semantics

We present the syntax and semantics of our augmented version of Timed CSP. Further details can be found in [19], where in particular we address the question of detecting livelocked processes (able to perform infinitely many internal transitions). In this paper, we only consider processes that are livelock-free.

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 n ranges over the non-negative integers $\mathbb{N}$. R denotes a (renaming) relation on Σ. The variable X is drawn from a fixed infinite set of process variables VAR.

Timed CSP terms are constructed according to the following grammar:

$$\begin{aligned} P ::= {} & \mathit{STOP} \mid \mathit{TIMESTOP} \mid a \longrightarrow P \mid a \xrightarrow{!} P \mid \mathit{SKIP} \mid \mathit{RANDOM} \mid \\ & P_1 \overset{n}{\triangleright} P_2 \mid P_1 \overset{n}{\lightning} P_2 \mid P_1 \Box P_2 \mid P_1 \sqcap P_2 \mid P_1 \underset{A}{\parallel} P_2 \mid P_1 \fatsemi P_2 \mid \\ & P \setminus A \mid P[R] \mid X \mid \mu X \centerdot P \ . \end{aligned}$$

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 ✓ at any time. *RANDOM* nondeterministically waits for a real-valued amount of time, and then becomes *SKIP*. $P \overset{n}{\triangleright} Q$ is the timeout process that initially offers to become P for n time units, after which it silently becomes Q if P has failed to communicate any visible event. $P \overset{n}{\lightning} Q$ is the interrupt process that behaves like P for the first n time units, and then silently starts behaving like Q (assuming P has not successfully terminated in the meantime). $P \sqcap Q$ denotes the nondeterministic choice between P and Q, whereas $P \Box Q$ represents the deterministic alternative. $P \Box Q$ is willing to behave either like P or like Q, 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 synchronise on all events in A, and to behave independently of each other with respect to all other events. $P \fatsemi Q$ is the sequential composition of P and Q: it denotes a process which behaves like P until P chooses to terminate (silently), at which point the process seamlessly starts to behave like Q. $P \setminus A$ is a process which behaves like P but with all communications in the set A hidden; the assumption of *maximal progress*, or *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 $a \ R \ b$. Lastly, the recursion $\mu X \centerdot P$ represents the unique solution to the equation $X = P$.

The requirement that all *delays* (parameters n in the terms $P_1 \overset{n}{\triangleright} P_2$, $P_1 \overset{n}{\lightning} P_2$) be integral is very benign—thanks to the possibility of scaling time units, we could have equally well required rational delays instead, with only minor modifications to our presentation.

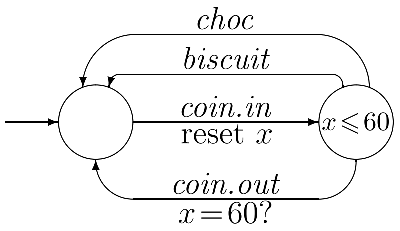
We write **TCSP** to denote the collection of closed terms of the language thus generated. (A term is closed if every occurrence of a variable X in it is within the scope of a μX operator).

We occasionally use the following derived constructs: if $S = \{P_1, \ldots , P_k\}$ is a finite set of processes, $\bigsqcap S$ represents $P_1 \sqcap \ldots \sqcap P_k$, 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 $\mu X \centerdot P$.

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 introduced 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 4).

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



We now sketch the dense-time denotational semantics of this augmented version of Timed CSP; a congruent operational semantics is also given in Appendix A. The denotational semantics is a hybrid mix of Reed and Roscoe's timed failures model [22,21,23,27] and refusal testing [20].

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 $(t, time)$, where *time* is a symbol indicating that time cannot pass. From now on, we abbreviate $\Sigma^{\checkmark} \cup \{time\}$ as $\Sigma^{\checkmark}_{time}$. We require in addition that refusals be time-bounded (the set of times associated with a refusal's timed events must be bounded).

A (timed) *refusal trace* is a 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 condition that timestamps be non-decreasing. The set of all refusal traces is written RT.

We interpret a refusal trace T 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. No refusal is however recorded while silent actions are on offer. The refusal of time itself is communicated to the experimenter via, say, the flashing of a red light.

Refusal traces are essentially linear-time behaviours that encapsulate a modicum of branching-time information. The primary motivation for considering refusal traces as opposed to simple timed traces is that the latter do not allow for a compositional model, as was shown in [21].

A notable feature of Timed CSP is the phenomenon of *point nondeterminism*, whereby distinct behaviours are nondeterministically enabled or disabled at the precise instant at which a silent transition (such as a timeout) occurs. Point nondeterminism, discussed at much greater length in [23], arises naturally in models of Timed CSP for compositionality reasons. Interestingly, one of its chief consequences in the present context is that it allows for digitisation analysis of dense-time liveness properties (such as timestop-freedom), which other modelling paradigms, such as timed automata, usually lack. We shall return to this point in Section 5.

One can define a compositional semantic map $\mathcal{R}_{\mathbb{R}}[\![\cdot]\!] : \mathbf{TCSP} \longrightarrow \mathcal{P}(RT)$ which associates with every process P its set of (dense-time) refusal traces $\mathcal{R}_{\mathbb{R}}[\![P]\!]$. This map can also be described through the operational semantics, as shown in Appendix A.

One can also compositionally define $\mathcal{R}_{\mathbb{Z}}[\![P]\!]$, the set of integral-time refusal traces 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 with integral endpoints. A *refusal token* is a set of the form $[t, t'] \times A$, $(t, t'] \times A$, $[t, t') \times A$, or $(t, t') \times A$, where $t, t' \in \mathbb{R}^+$ are the *endpoints* of the refusal token, and $A \subseteq \Sigma^{\checkmark}_{time}$ are the events refused over the corresponding interval.

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.

3 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 [15] 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 traces or refusal traces, and according to whether time is dense or discrete (integral):

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

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 that they are captured by Timed CSP processes. For simplicity, we have ignored the possibility of global successful termination (communication of ✓'s).

Four of the specifications are given their corresponding MTL or TCTL formulas as names, but no knowledge of these logics is required or assumed.

Safe reachability ($\Box\neg a$): 'The event a is never performed.' According to [10], 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 $\mathit{RUN}_{\Sigma-\{a\}}$. Here $\mathit{RUN}_B = \Box\{b \longrightarrow \mathit{RUN}_B \mid b \in B\}$. RUN_B can perform any trace containing only events in B.

Constant availability ($\forall\Box a$): 'The event a is never refused.' The process $\mathit{LIVE} = \Big(\mathit{RANDOM} \mathbin{;} (\bigsqcap\{b \xrightarrow{!} \mathit{LIVE} \mid b \in \Sigma\} \sqcap \mathit{TIMESTOP})\Big) \mathbin{|||} a \longrightarrow \mathit{LIVE}$ captures this refusal trace liveness specification.

Timestop-freedom (TSF): 'The process never exhibits timestops.' The process $\mathit{TSF} = \mathit{RANDOM} \mathbin{;} (\bigsqcap\{a \xrightarrow{!} \mathit{TSF} \mid a \in \Sigma\} \sqcap \mathit{STOP})$ captures this refusal trace liveness specification. TSF is the most nondeterministic process which has no timestops.

[12] lists the two specifications that follow as the most commonly encountered in practice; note that safe reachability is essentially a special case of bounded invariance.

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.

This trace specification can be captured 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, as we now demonstrate.

We employ digitisation techniques [12,16,17] as a means of verifying that a process meets its specification. (A brief review of digitisation is presented in Appendix B.) The following theorem, extending a result of [16,17], is central to our approach:

Theorem 3.1 *Any $P \in$ **TCSP** is closed under refusal trace (and hence also trace) digitisation.*

Proposition 3.2 *Safe reachability, bounded invariance, and bounded response are closed under inverse trace digitisation [12]. Timestop-freedom and constant availability are closed under inverse refusal trace digitisation.*

Corollary 3.3 *Let $P \in$ **TCSP** be a Timed CSP process. Then*

$$P \vDash^{\mathcal{T}}_{\mathbb{R}} \Box\neg a \Leftrightarrow P \vDash^{\mathcal{T}}_{\mathbb{Z}} \Box\neg a$$

$$P \vDash^{\mathcal{R}}_{\mathbb{R}} \forall \Box a \Leftrightarrow P \vDash^{\mathcal{R}}_{\mathbb{Z}} \forall \Box a$$

$$P \vDash^{\mathcal{R}}_{\mathbb{R}} TSF \Leftrightarrow P \vDash^{\mathcal{R}}_{\mathbb{Z}} TSF$$

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

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

The right-hand side discrete checks can all be performed on the model checker FDR; see [16,17] for details.

4 Closed Timed Automata

We define the class of *closed timed automata*. These are essentially the timed safety automata of [13] with exclusively *closed* invariant and enabling 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, this restriction appears to be rather benign in practice, an opinion shared by several researchers (see, e.g., [7]).

This class of timed automata corresponds exactly to 'finite-state' Timed CSP processes, in a sense which we make precise shortly. Since the semantics of Timed CSP is based on finite behaviours, we must give up Alur and Dill's Büchi acceptance conditions [4], and instead consider every state to be accepting. To simplify our exposition, we assume that automata cannot terminate successfully (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.

Definition 4.1 A *closed timed automaton* is a tuple $(\Sigma, S, S_0, C, E, \mathsf{inv})$, where

- Σ is a finite alphabet,
- S is a finite set of states,
- $S_0 \subseteq S$ is a set of start states,
- C is a finite set of clocks,
- $E \subseteq S \times S \times \Sigma \times \mathcal{P}(C) \times F_C$ is a set of transitions, and
- $\mathsf{inv} : S \longrightarrow F_C$ specifies state invariant constraints.

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

The dense-time refusal trace semantics $\mathcal{R}_{\mathbb{R}}[\![A]\!]$ of a timed automaton A can now be defined in the standard way; we have relegated the precise details to Appendix C.

The sets $\mathcal{T}_{\mathbb{R}}[\![A]\!]$, $\mathcal{R}_{\mathbb{Z}}[\![A]\!]$, and $\mathcal{T}_{\mathbb{Z}}[\![A]\!]$, representing respectively the dense-time traces of A, the integral-time refusal 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.

5 Timed Automata as Timed CSP Processes (and Back)

Given any timed automaton $A \in \mathbf{CTA}$, we construct two corresponding Timed CSP processes $P_{\mathbb{R}}^{A}$ and $P_{\mathbb{Z}}^{A}$, capturing respectively the dense-time and integral-time behaviours of A. Our constructions use some ideas introduced in [9].

We begin by giving the construction of $P_{\mathbb{R}}^{A}$. Let $A = (\Sigma, S, S_0, C, E, \mathsf{inv})$ be an automaton with k clocks $x_1, x_2, \ldots, x_k \in C$. We build $P_{\mathbb{R}}^{A}$ as the parallel composition of $k+2$ processes: a network *CLOCKS* of k processes for the clocks, a process *REGIONS* to mimic the clock regions graph, and a process *CONTROL* for the discrete state controller.

For each clock $x_i \in C$ of A, let c_{x_i} be the largest constant that x_i is ever compared to in the enabling and invariant constraints of A. We assume the reader is familiar with the *clock regions construction* [2,3], which partitions $(\mathbb{R}^+)^k$ (the space of valuations of clocks) into $k! \cdot 2^k \cdot \prod_{1 \leqslant i \leqslant k}(2c_{x_i} + 2)$ or fewer equivalence classes. Two clock interpretations lie in different equivalence classes if either they differ in the integral parts of the readings of a clock x_i (and one of these numbers is less than $c_{x_i} + 1$), or if they differ in the ordering of the fractional parts of all the clocks.

Each equivalence class, or clock region, r is modelled 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 *CONTROL*; upon entering a new discrete state, *CONTROL* can thus check whether or not the invariant constraint is satisfied. Now suppose that r' is the region immediately following r temporally. (In the case of 'extremal' regions r, no such r' exists, and the next sentence does not apply.) 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*$'$. The command *switch*.r' is initiated by *CLOCKS*, and also requires the participation of *CONTROL*, as a means to enforce the state 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 $a \in \Sigma$, the region-process *REGr* is always willing to accept the 'r-annotated event' $r.a$. Although this latter communication is external (not hidden), a subsequent global renaming operation restores all such communications to their nominal values (the simple event a in this case). The composite clock regions graph process is denoted *REGIONS*.

Each clock x_i is modelled by a process CLx_i. This process can be in any of the $2c_{x_i}+2$ following states (represented as disjoint subsets of $\mathbb{R}^+$): $\{0\}, (0,1), \{1\}, (1,2), \ldots, (c_{x_i}-1, c_{x_i}), \{c_{x_i}\}, (c_{x_i}, \infty)$. CLx_i switches from state to state by means of the interrupt operator; it spends unit-duration periods in bounded 'interval' states, and null-duration periods in 'singleton' states. Prior to entering a new state, it offers as signals a choice of all the events of the form $switch.r$, where r is any region compatible with the value of x_i in the new state. Moreover, all the clocks for which r is temporally a 'border' region must participate in the event $switch.r$. CLx_i is also 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 $CLOCKS$.

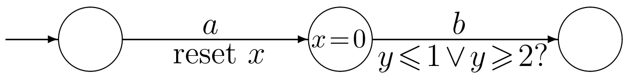
Any clock constraint $\sigma \in F_C$ can be identified with a subset $\{r \mid r \vDash \sigma\}$ of clock regions. Whenever the automaton A communicates an event a under a particular clock valuation ν, the process $P^A_{\mathbb{R}}$ is meant to communicate a corresponding region-annotated event $r.a$, where r is the region corresponding to ν.[4] Recall that this event is subsequently renamed to a at the outermost level.

The discrete controller is modelled by a process $CONTROL$, which mimics the various states in S of the automaton. Initially, $CONTROL$ nondeterministically begins in one of the start states in S_0. Upon entering a new state $s \in S$, the first thing $CONTROL$ does is signal to $REGIONS$ a choice of events of the form $query.r$, where $r \in \mathsf{inv}(s)$. If $REGIONS$ cannot synchronise on any of these events, this means a state has been reached in which the invariant constraint is violated, and a timestop automatically ensues. Otherwise, while in state s the discrete controller is always willing to accept events of the form $switch.r$, as long as $r \in \mathsf{inv}(s)$. For every edge $e = (s, s', a, D, \sigma)$, $CONTROL$, while in state s, continuously offers a choice of region-annotated events of the form $r.a$, where $r \in \sigma$. If any of these transitions is accepted, then $CONTROL$ immediately proceeds to signal the events $reset.x$, for every clock $x \in D$ to be reset. Finally, it enters the new state s', subject to the initial satisfaction check of s''s invariant constraint.

$CONTROL$, $REGIONS$, and $CLOCKS$ are combined in parallel and required to synchronise on all appropriate events. Events on internal channels are then hidden, and, 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 turns out that $P^A_{\mathbb{R}}$ has exactly the same timed traces as A, but not quite the same refusals: because of point nondeterminism, whenever A refuses a set of events over an open time interval (t, t'), $P^A_{\mathbb{R}}$ is able to refuse the same set over $[t, t')$. Interestingly, it is precisely point nondeterminism which makes Timed CSP processes closed under *refusal* trace digitisation. Closed timed automata, on the other hand, do not have this property, as can be seen from examining the timed automaton A below:

[4] The infinitesimal uncertainty introduced by point nondeterminism however imposes the caveat that a given clock interpretation may belong to two different regions, at least as far as $P^A_{\mathbb{R}}$ is concerned.



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

Having noticed such phenomena, Bošnački comments in [8] that digitisation techniques appear to be inadequate to handle the requirement of timestop-freedom. Corollary 5.2 shows that this is not the case (although our region-graphs-based method is certainly no more efficient than the algorithm of [13]).

The following result establishes that the expressiveness of Timed CSP is at least that of closed timed automata; the 'left-closure' operator $\overleftarrow{(\cdot)}$ is defined in Appendix C.

Theorem 5.1 *For any timed automaton $A \in$ **CTA**, $\mathcal{R}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!] = \overleftarrow{\mathcal{R}_{\mathbb{R}}[\![A]\!]}$.*

In particular, $\mathcal{T}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!] = \mathcal{T}_{\mathbb{R}}[\![A]\!]$. Thus closed timed automata are closed under trace digitisation.

Corollary 5.2 *Let $A \in$ **CTA** be a timed automaton. The following are equivalent:*

(i) *A is timestop-free ($A \vDash^{\mathcal{R}}_{\mathbb{R}} TSF$).*

(ii) *$P^A_{\mathbb{R}}$ is timestop-free ($P^A_{\mathbb{R}} \vDash^{\mathcal{R}}_{\mathbb{R}} TSF$).*

(iii) *The integral refusal traces of $P^A_{\mathbb{R}}$ are timestop-free ($P^A_{\mathbb{R}} \vDash^{\mathcal{R}}_{\mathbb{Z}} TSF$).*

The last of these is a discrete check which can be performed on FDR.

A legitimate question is whether Timed CSP is any more expressive than closed timed automata. Since Timed CSP is Turing-complete, the answer must be affirmative; however, we find that both formalisms are equally expressive if we restrict ourselves to *finite-state* processes, as defined below.

Any $P \in$ **TCSP** has an associated labelled transition system, calculated using the operational semantics of Appendix A. The sub-labelled transition system of P in which all evolutions have unit duration is the *discrete* labelled transition system of P. We say that P is *finite-state* if its discrete labelled transition system is finite.

Theorem 5.3 *Let $P \in$ **TCSP** be a finite-state process. Then there exists a closed timed automaton $A \in$ **CTA** such that $\mathcal{R}_{\mathbb{R}}[\![P]\!] = \mathcal{R}_{\mathbb{R}}[\![P^A_{\mathbb{R}}]\!]$.*

This theorem significantly improves a previous result of Jackson [14], in that our 'finite-state' criterion is clearly both necessary and sufficient. In *op. cit.*, severe syntactic restrictions are imposed on the Timed CSP syntax in order to derive the corresponding (weaker) result.

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 $\{(j_1, j_2, \ldots, j_k)\}$, i.e., integral singletons in $(\mathbb{R}^+)^k$. Note that the number of regions is then $\prod_{1 \leqslant i \leqslant k}(c_{x_i} + 2)$. 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 $P^A_{\mathbb{Z}}$.

Theorem 5.4 *For any timed automaton $A \in$ **CTA**, $\mathcal{T}_{\mathbb{Z}}[\![P^A_{\mathbb{Z}}]\!] = \mathcal{T}_{\mathbb{Z}}[\![A]\!]$.*

The significance of this result comes from the applicability of digitisation techniques to the verification problem, as detailed in Section 3.

We conclude by noting that in general, it does not always seem necessary to construct the full regions graph to capture the dense-time behaviours of a given timed automaton. Likewise, even for discrete behaviours, it is not always necessary to consider every lattice point. The minimal graph that needs to be constructed depends on both the specification to be checked and on the set of invariant and enabling constraints of the automaton. How to construct the minimal discrete controller seems to be an interesting topic for further research.

6 Conclusion and Future Work

We have characterised the precise expressive power of (finite-state) Timed CSP as that of closed timed automata—timed safety automata [13] with closed invariant and enabling clock constraints.

We have also shown that Timed CSP 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 branching-time liveness properties of timestop-freedom and constant availability.

We have established that Timed CSP processes are closed under refusal trace digitisation [16,17], and that since all the specifications listed above are closed under inverse (refusal) trace digitisation, the corresponding verification problems discretise and can be handled by the model checker FDR.

A number of questions remain open. One concerns the expressiveness of Timed CSP as a *specification* formalism. Following the lead of [15] 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. Another interesting project would be to develop techniques to minimise the number of discrete states needed to capture the behaviours of a given timed automaton. Lastly, although deciding whether a process or timed automaton is closed under inverse digitisation is undecidable [18], it would be very useful to devise simple criteria that would guarantee closure under inverse digitisation.

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

References

[1] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of LICS 90*, pages 414–425. IEEE Computer Society Press, 1990.

[3] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[4] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[5] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[6] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[7] E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In *Proceedings of CONCUR 98*, volume 1466, pages 470–484. Springer LNCS, 1998.

[8] D. Bošnački. Digitization of timed automata. In *Proceedings of FMICS 99*, 1999.

[9] R. Chapman and M. Goldsmith. Translating timer automata to TCSP. Formal Systems Design and Development, Inc., 1995.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[11] J. Davies. *Specification and Proof in Real-Time Systems*. PhD thesis, Oxford University, 1991.

[12] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of ICALP 92*, volume 623, pages 545–558. Springer LNCS, 1992.

[13] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[14] D. M. Jackson. *Logical Verification of Reactive Software Systems*. PhD thesis, Oxford University, 1992.

[15] M. Leuschel, T. Massart, and A. Currie. How to make FDR Spin: LTL model checking using refinement. In *Proceedings of FME 01*, volume 2021, pages 99–118. Springer LNCS, 2001.

[16] J. Ouaknine. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems*. PhD thesis, Oxford University, 2001. Technical report PRG-RR-01-06.

[17] J. Ouaknine. Digitisation and full abstraction for dense-time model checking. In *Proceedings of TACAS 02*, volume 2280, pages 37–51. Springer LNCS, 2002.

[18] J. Ouaknine and J. B. Worrell. Some decidability and undecidability results for timed automata. Submitted, 2002. `www.math.tulane.edu/~joelo`.

[19] J. Ouaknine and J. B. Worrell. Towards specification as refinement in timed systems. In *Proceedings of AVoCS 02*. The University of Birmingham, 2002.

[20] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.

[21] G. M. Reed. *A Mathematical Theory for Real-Time Distributed Computing*. PhD thesis, Oxford University, 1988.

[22] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *Proceedings of ICALP 86*, pages 314–323. Springer LNCS, 1986. *Theoretical Computer Science*, 58:249–261.

[23] G. M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *Theoretical Computer Science*, 211:85–127, 1999.

[24] S. A. Schneider. *Correctness and Communication in Real-Time Systems*. PhD thesis, Oxford University, 1989.

[25] S. A. Schneider. Using CSP with Z in the mine pump case study. Unpublished, 1994.

[26] S. A. Schneider. An operational semantics for Timed CSP. *Information and Computation*, 116:193–213, 1995.

[27] S. A. Schneider. *Concurrent and Real Time Systems: the CSP approach*. John Wiley, 2000.

A Timed CSP Operational Semantics

The contents and style of this appendix are similar to [26]. We present a collection of inference rules which allow us to assign to any Timed CSP process a set of dense-time *executions*.

To this end, we must relax the syntactic requirement that all delays (parameters n in the terms $P_1 \overset{n}{\rhd} P_2$, $P_1 \overset{n}{\lightning} P_2$) be integral, and allow arbitrary non-negative real numbers instead (written t in place of n). This expanded collection of closed terms is written $TCSP$.

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). $P \overset{\gamma}{\not\longrightarrow}$ means that P cannot possibly do a γ at that particular time. $P \overset{t}{\rightsquigarrow} P'$ means that P can become P' simply by virtue of letting t units of time elapse, where $t \in \mathbb{R}^+$.

$P \not\rightsquigarrow$ means that P can let no strictly positive amount of time elapse. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_1 \not\xrightarrow{\checkmark}}{P_1 \mathbin{;} P_2 \overset{t}{\rightsquigarrow} P_1' \mathbin{;} P_2} \qquad \frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \mathbin{;} P_2 \xrightarrow{\tau} P_2} \qquad \frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \mathbin{;} P_2 \xrightarrow{\gamma} P_1' \mathbin{;} P_2} \; [\, \gamma \neq \checkmark \,]$$

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

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

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

$$\frac{}{\mu\, X \centerdot P \xrightarrow{\tau} [(\mu\, X \centerdot P)/X]P} \; .$$

Note that there are no rules for $TIMESTOP$.

For $P \in TCSP$, we define the set of events immediately refused by P, $\mathsf{ref}(P) \subseteq \Sigma^{\checkmark}_{time}$, as follows: if $P \xrightarrow{\tau}$, then $\mathsf{ref}(P) = \emptyset$. Otherwise, for $a \in \Sigma^{\checkmark}$, $a \in \mathsf{ref}(P) \Leftrightarrow P \not\xrightarrow{a}$, and $time \in \mathsf{ref}(P) \Leftrightarrow P \not\rightsquigarrow$. (Recall that $time$ is a special 'event' not belonging to $\Sigma^{\checkmark}$.)

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 $P_i \xrightarrow{\gamma} P_{i+1}$ (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 $\mathsf{exec}(P)$.

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* $T \frown\!\!\!\!\cdot\; T' \mathrel{\widehat{=}} \langle \aleph_0, (t_1, a_1), \ldots, (t_k, \aleph_k), \aleph_k \cup \aleph_0', (t_1', a_1'), \ldots, \aleph_l' \rangle$.

The operator $\frown$, on the other hand, denotes simple sequence concatenation.

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.

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

$$\begin{aligned}
\mathsf{rt}(P) &\mathrel{\widehat{=}} \langle \{0\} \times \mathsf{ref}(P) \rangle \\
\mathsf{rt}((P \xrightarrow{\tau})^\frown e) &\mathrel{\widehat{=}} \mathsf{rt}(e) \\
\mathsf{rt}((P \xrightarrow{a})^\frown e) &\mathrel{\widehat{=}} \langle \{0\} \times \mathsf{ref}(P), (0, a) \rangle^\frown \mathsf{rt}(e) \\
\mathsf{rt}((P \overset{t}{\rightsquigarrow})^\frown e) &\mathrel{\widehat{=}} \langle [0, t) \times \mathsf{ref}(P) \rangle \frown\!\!\!\!\cdot\; (\mathsf{rt}(e) + t) \; .
\end{aligned}$$

We define a partial order $\prec$ on refusal traces, which orders refusal traces according to how much information they contain. If $T = \langle \aleph_0, (t_1, a_1), \ldots , \aleph_k\rangle$ and $T' = \langle \aleph'_0, (t'_1, a'_1), \ldots , \aleph'_l\rangle$ are refusal traces, we let $T' \prec T$ in case $l \leqslant k \wedge \aleph'_0 \subseteq \aleph_0 \wedge \forall(1 \leqslant i \leqslant l) \centerdot a'_i = a_i \wedge \aleph'_i \subseteq \aleph_i$. This definition gives rise to an operator $\downarrow$: for P a set of refusal traces, $\downarrow P$ is the smallest $\prec$-downward-closed set containing P.

The congruence theorem reads:

Theorem A.1 *For any $P \in \mathbf{TCSP}$, $\mathcal{R}_{\mathbb{R}}[\![P]\!] = \downarrow \mathsf{rt}(\mathsf{exec}(P))$.*

B Digitisation

Digitisation techniques were first introduced in [12], and later extended from traces to refusal traces in [16,17]. We review the main points, adapted to the present framework.

Let $t \in \mathbb{R}^+$, and let $0 \leqslant \varepsilon \leqslant 1$ be a real number. Decompose t into its integral and fractional parts, thus: $t = \lfloor t \rfloor + t'$. If $t' < \varepsilon$, let $[t]_\varepsilon \mathrel{\widehat{=}} \lfloor t \rfloor$, otherwise let $[t]_\varepsilon \mathrel{\widehat{=}} \lceil t \rceil$.

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

Let $A \subseteq \mathbb{R}^+ \times \Sigma^{\checkmark}_{time}$ be a refusal. Write $A = \bigcup_{i \in \mathcal{I}} \{I_i \times A_i\}$, where each I_i is an open, half-open, or closed interval with endpoints $u_i, v_i \in \mathbb{R}^+$, and $A_i \subseteq \Sigma^{\checkmark}_{time}$. Assume moreover that this representation is *maximal* in that the intervals I_i are as large as possible. We then define $[A]_\varepsilon$ by pointwise application to the endpoints u_i, v_i of the intervals I_i.

Naturally, this extends $[\cdot]_\varepsilon$ to refusal traces, and hence sets thereof, in the obvious way. The overloading of the notation causes no confusion since the arguments of $[\cdot]_\varepsilon$ are of disjoint types.

Definition B.1 A set P of (refusal) traces is *closed under (refusal) trace digitisation* if, for any $0 \leqslant \varepsilon \leqslant 1$, $[P]_\varepsilon \subseteq P$.

A set S of (refusal) traces is *closed under inverse (refusal) trace digitisation* if, whenever a (refusal) trace s is such that $[s]_\varepsilon \in S$ for all $0 \leqslant \varepsilon \leqslant 1$, then $s \in S$.

For P and S Timed CSP processes, the above definitions apply respectively to $\mathcal{T}_{\mathbb{R}}[\![P]\!]$ ($\mathcal{R}_{\mathbb{R}}[\![P]\!]$) and $\mathcal{T}_{\mathbb{R}}[\![S]\!]$ ($\mathcal{R}_{\mathbb{R}}[\![S]\!]$).

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

The main verification result is as follows:

Theorem B.2 *Let P be a set of (refusal) traces closed under (refusal) trace digitisation, and let S be a set of (refusal) traces closed under inverse (refusal) trace digitisation. Then $P \subseteq S$ if and only if $\mathbb{Z}(P) \subseteq \mathbb{Z}(S)$.*

C Closed Timed Automata Semantics

We assume a timed automaton $A = (\Sigma, S, S_0, C, E, \mathsf{inv}) \in \mathbf{CTA}$. We define the set of refusal traces of A in a manner similar to [4,13].

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 $[\text{reset } D]\nu$ be the clock interpretation which evaluates clocks in D to 0 and agrees with ν on clocks outside of D.

Given a state s and a clock interpretation ν, let $\mathsf{ref}(s,\nu) \subseteq \mathbb{R}^+ \times \Sigma^{\checkmark}_{time}$ be defined as follows: if $\nu \nvDash \mathsf{inv}(s)$, then $\mathsf{ref}(s,\nu) \mathrel{\widehat{=}} \{0\} \times \Sigma^{\checkmark}_{time}$. Otherwise, for $t \in \mathbb{R}^+$ and $a \in \Sigma^{\checkmark}$, we let $(t,a) \in \mathsf{ref}(s,\nu)$ if, for all $\delta \in [0,t]$, $\nu + \delta \vDash \mathsf{inv}(s)$, and also if there is no s-originating, a-labelled, and σ-enabled transition $e \in E$ (i.e., $e = (s, -, a, -, \sigma)$) with $\nu + t \nvDash \sigma$. Lastly, we let $(t, \mathit{time}) \in \mathsf{ref}(s,\nu)$ if t is the largest finite real number with the property that, for all $\delta \in [0,t]$, $\nu + \delta \vDash \mathsf{inv}(s)$.

A *run* over A is a finite sequence $e = (s_0,t_0,\nu_0) \xrightarrow{\alpha_1} (s_1,t_1,\nu_1) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (s_n,t_n,\nu_n)$, where each state $s_i \in S$, the t_i's $\in \mathbb{R}^+$ are non-decreasing, each $\nu_i : C \longrightarrow \mathbb{R}^+$ is a clock interpretation, each transition $\alpha_i = (s_{i-1}, s_i, a_i, D_i, \sigma_i) \in E$, and all of which are subject to the following conditions:

(i) $s_0 \in S_0$, $t_0 = 0$, and $\nu_0(x) = 0$ for all $x \in C$.

(ii) For all $0 \leqslant i \leqslant n-1$, for all $\delta \in [t_i, t_{i+1}]$, $\nu_i + \delta \vDash \mathsf{inv}(s_i)$.

(iii) For all $0 \leqslant i \leqslant n-1$, $\nu_i + (t_{i+1} - t_i) \vDash \sigma_{i+1}$.

(iv) For all $0 \leqslant i \leqslant n-1$, $\nu_{i+1} = [\text{reset } D_i](\nu_i + (t_{i+1} - t_i))$.

The set of runs of A is written $\mathsf{run}(A)$.

Given a run $e = (s_0,t_0,\nu_0) \xrightarrow{\alpha_1} (s_1,t_1,\nu_1) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (s_n,t_n,\nu_n)$, we produce an associated refusal trace $\mathsf{rt}(e) = \langle \aleph_0, (t_1,a_1), \aleph_1, (t_2,a_2), \ldots, (t_n,a_n), \aleph_n \rangle$ as follows:[5] The a_i's are extracted from the α_i's, and for $0 \leqslant i \leqslant n-1$, we let $\aleph_i \mathrel{\widehat{=}} (\mathsf{ref}(s_i,\nu_i) + t_i) \cap ([t_i, t_{i+1}] \times \Sigma^{\checkmark}_{time})$. Lastly, we let $\aleph_n \mathrel{\widehat{=}} \mathsf{ref}(s_n,\nu_n) + t_n$.

We can thus derive the set of (dense-time) refusal traces associated with a timed automaton A: $\mathcal{R}_{\mathbb{R}}[\![A]\!] \mathrel{\widehat{=}} \downarrow\mathsf{rt}(\mathsf{run}(A))$.

Finally, we define the left-closure operator $\overleftarrow{(\cdot)}$ as follows:

Let $R \subseteq \mathbb{R}^+ \times \Sigma_{time}$ be a refusal. Define a new refusal $\overleftarrow{R} \supseteq R$ such that, for any $t \in \mathbb{R}^+$ and $a \in \Sigma_{time}$, whenever $\{\delta_i\} \subseteq \mathbb{R}^+$ is a set of positive real numbers with infimum 0, if $\{(t + \delta_i, a)\} \subseteq R$, then $(t,a) \in \overleftarrow{R}$. This definition extends to refusal traces by pointwise application to the refusal components of the refusal trace.

For P a set of refusal traces, let $\overleftarrow{P} \mathrel{\widehat{=}} \downarrow\{\overleftarrow{T} \mid T \in P\}$.

[5] For convenience, we allow the last refusal of $\mathsf{rt}(e)$ to be unbounded.