Digitisation and Full Abstraction for Dense-Time Model Checking[*]

Joël Ouaknine

Department of Mathematics, Tulane University,
New Orleans LA 70118, USA
joelo@math.tulane.edu

Abstract. We study the digitisation of dense-time behaviours of timed processes, and show how this leads to exact verification methods for a large class of dense-time specifications. These specifications are all *closed under inverse digitisation*, a robustness property first introduced by Henzinger, Manna, and Pnueli (on timed traces), and extended here to timed failures, enabling us to consider liveness issues in addition to safety properties. We discuss a corresponding model checking algorithm and show that, in many cases, automated verification of such dense-time specifications can in fact be directly performed on the model checker FDR (a commercial product of Formal Systems (Europe) Ltd.). We illustrate this with a small case study (the railway level crossing problem). Finally, we show that integral—or digitised—behaviours are fully abstract with respect to specifications closed under inverse digitisation, and relate this to the efficiency of our model checking algorithm.

1 Introduction

Real-time systems appear in an increasingly large number of applications, from kitchen appliances to nuclear power, telecommunications, aeronautics, and so on. In many instances, it is crucial that such systems behave exactly as they were intended to, lest catastrophic consequences ensue. For this reason, real-time systems have elicited a large amount of research in recent years.

Timed systems can broadly be divided into two subclasses, depending on whether time is modelled in a *discrete* or *dense* fashion. Early research focussed mostly on the former, as the verification techniques already in circulation could readily be extended to this case. There are however a number of drawbacks associated with discrete modelling of time, and chief among them is the obvious limit on the accuracy of the analysis. Dense-time systems (typically modelled over the reals), on the other hand, have infinite (usually uncountable) state spaces, ruling out the direct application of techniques such as model checking.

One of the earliest attempts to circumvent this problem was the technique of *timewise refinement* [24, 22, 26]. The crux of this method consists in removing

[*] This research was supported in part by the U.S. Office of Naval Research under contract N00014-95-1-0520 and by the Fonds FCAR (Québec).

all timing information from a timed system, keeping only the relative order in which state changes occur. In this way one obtains an untimed (discrete) system, which can then be handled by standard model checking algorithms. The obvious disadvantage of this technique is that the class of specifications that can be soundly verified with it is rather restricted; on the other hand, timewise refinement is usually extremely efficient in those cases in which it can be applied.

At the other end of the spectrum we find the technique of *region graphs* [1, 2]. This consists in defining an equivalence relation on the state space of a timed system, giving rise to a finite number of partitions, or *regions*. Points within a given region are indistinguishable insofar as the verification of specifications (usually expressed as TCTL formulas) is concerned; this observation yields a model checking procedure. The advantage of this approach is the large class of specifications it is able to handle. On the other hand, it tends to be computationally very expensive, as regions partition the state space extremely finely.

Many intermediate and related techniques have since appeared, of which the following is only a partial list: [15, 13, 4, 5, 3, 14, 16, 28, 12, 11, 29, 19]. Some are concerned with discretisation methods or symbolic analysis, while others offer various trade-offs (space economy vs. runtime efficiency, or runtime efficiency vs. wideness of applicability, etc.). There have also been a small number of model checking tools developed for timed systems, such as Cospan [6], UppAal [7], Kronos [8], and HyTech [10].

Our interest lies in the connection between discrete-time and dense-time paradigms for modelling timed systems, and in particular the scope and limitations of applying discrete-time methods to dense-time verification problems. Henzinger, Manna, and Pnueli studied this question in [13], and concluded that one must restrict one's attention both to systems *closed under digitisation* and to specifications *closed under inverse digitisation*. Their analysis focussed on *timed transition systems* and considered a *timed trace* semantics. In this paper, we extend their results to *timed failures*, enabling us to address liveness issues in addition to safety properties.

The framework we have chosen is Reed and Roscoe's Timed CSP [20, 21, 27, 25]. Timed CSP is a well-known process algebra for timed systems and is a natural extension of CSP, which itself has been used to model timed systems discretely [23]. In addition, Timed CSP is endowed with both denotational and operational semantics, so that questions may be studied from both viewpoints. For reasons of space, however, the work we present here focusses on the denotational side; one may find an operational account of it in [18].

One of our main results is the *digitisation lemma*, which states that all Timed CSP processes are closed under digitisation. On the specification side, we find indeed that specifications closed under inverse digitisation are verifiable, and moreover that this is a hard restriction which essentially cannot be relaxed. Another very interesting result is that integral behaviours are in fact *fully abstract* with respect to the class of specifications under consideration; in other words, integral behaviours contain precisely the right amount of information required for our verification aims, without any junk.

These results lead us to a model checking algorithm, which can in fact be implemented on FDR [23], a CSP model checker. We illustrate this in a small case study.

2 Timed CSP Syntax and Conventions

Let Σ be a finite set of *events*. In the notation below, we have $a \in \Sigma$ and $A \subseteq \Sigma$. The parameter n ranges over the non-negative integers $\mathbb{N}$. f represents a function $f : \Sigma \longrightarrow \Sigma$. The variable X is drawn from a fixed infinite set of process variables $\mathit{VAR} \mathrel{\widehat{=}} \{X, Y, Z, \ldots\}$.

Timed CSP terms are constructed according to the following grammar:

$$\begin{aligned} P := {} & \mathit{STOP} \mid a \longrightarrow P \mid a : A \longrightarrow P(a) \mid \mathit{SKIP} \mid \mathit{WAIT}\ n \mid P_1 \overset{n}{\triangleright} P_2 \mid \\ & P_1 \mathbin{\Box} P_2 \mid P_1 \sqcap P_2 \mid P_1 \underset{A}{\|} P_2 \mid P_1 \mathbin{|||} P_2 \mid P_1 \mathbin{;} P_2 \mid P \setminus A \mid \\ & f^{-1}(P) \mid f(P) \mid X \mid \mu X \centerdot P \ . \end{aligned}$$

These terms have the following intuitive interpretations:

- STOP is the deadlocked, stable process which is only capable of letting time pass.
- $a \longrightarrow P$ initially offers at any time to engage in the event a, and subsequently behaves like P. The general prefixed process $a : A \longrightarrow P(a)$ is initially prepared to engage in any of the events $a \in A$, at the choice of the environment, and thereafter behave like $P(a)$; this corresponds to STOP when $A = \emptyset$.
- SKIP intuitively corresponds to the process $\checkmark \longrightarrow \mathit{STOP}$, where the event $\checkmark$ represents successful termination.
- $\mathit{WAIT}\ n$ is the process which idles for n time units, and then becomes SKIP.
- $P \overset{n}{\triangleright} Q$ is a 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.
- $P \sqcap Q$ represents the nondeterministic (or internal) choice between P and Q. Which of these two processes $P \sqcap Q$ becomes is independent of the environment, and how the choice is resolved is considered to be outside the domain of discourse.
- $P \mathbin{\Box} Q$, on the other hand, denotes a process which is willing to behave either like P or like Q, at the choice of the environment. This decision is taken on the first visible event that is communicated, and is nondeterministic only if this initial event, when it occurs, is possible for both P and Q.
- The parallel composition $P_1 \underset{A}{\|} P_2$ of P_1 and P_2, over the interface set A, requires P_1 and P_2 to agree and synchronise on all events in A, and to behave independently of each other with respect to all other events. The interleaving $P_1 \mathbin{|||} P_2$ corresponds to parallel composition over an empty interface.

- $P \fatsemi Q$ corresponds to the sequential composition of P and Q: it denotes a process which behaves like P until P successfully terminates (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 (made invisible to the environment); 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 processes $f^{-1}(P)$ and $f(P)$ derive their behaviours from those of P in that, whenever P can perform an event a, $f^{-1}(P)$ can engage in any event b such that $f(b) = a$, whereas $f(P)$ can perform $f(a)$.
- The process variable X has no intrinsic behaviour of its own, but can imitate any process P as part of a recursion—see below.
- The recursion $\mu X \centerdot P$ represents a process which behaves like P but with every free occurrence of X in P (recursively) replaced by $\mu X \centerdot P$. Semantically, this corresponds to the unique solution to the equation $X = P$. Note that the variable X here usually appears freely within P's body.

Let us write **TCSP** to denote the collection of closed terms of the language thus generated, i.e., terms in which every occurrence of a variable X is within the scope of a μX operator. We refer to these terms as *processes*.

Note our requirement that all *delays* (parameters n in the terms $\mathit{WAIT}\ n$ and $P_1 \overset{n}{\rhd} P_2$) be integral. This restriction is both necessary (since otherwise the main questions considered here become theoretically intractable) and essentially harmless, because of the freedom to scale time units. We could equivalently have required *rational* delays, as many authors do.

We occasionally use the following derived constructs: abbreviating $a \longrightarrow \mathit{STOP}$ as simply $\dot{a}$, and writing $a \xrightarrow{n} P$ instead of $a \longrightarrow \mathit{WAIT}\ n \fatsemi P$. We also tend to express recursions by means of the equational notation $X = P$, rather than the functional $\mu X \centerdot P$ prescribed by the definition.

Lastly, some mild additional technical syntactic restrictions, which we omit here, are required to ensure that processes are *non-Zeno*, or in other words that they never force time to *converge*. The reader may find more detailed discussion of this point in [17, 21, 27].

3 Dense-Time Modelling, Specification, and Verification

The standard denotational semantics for Timed CSP is Reed and Roscoe's *timed failures model*, $\mathcal{M}_{\mathbb{R}}$, a brief synopsis of which we now present. It comes equipped with a compositional[1] evaluation map $\mathcal{R}[\![\cdot]\!] : \mathbf{TCSP} \longrightarrow \mathcal{M}_{\mathbb{R}}$. This model allows one to assign dense-time (in fact, continuous-time) interpretations to Timed CSP processes. References include [20, 21, 27].

[1] By *compositional*, we mean that the value of a compound expression can be calculated from the values of its constituting subexpressions.

Timed failures are pairs $(s, \aleph)$, with s a *timed trace* and $\aleph$ a *timed refusal*. A *timed trace* is a finite sequence of *timed events* $(t, a) \in \mathbb{R}^+ \times \Sigma$, with the time values in non-decreasing order. Notationally, we enclose the elements of a timed trace in angled brackets, e.g., $\langle (0, a), (3.2, b), (3.2, a), (4.76, a) \rangle$, etc. A *timed refusal* is a set of timed events consisting of a finite union of *refusal tokens* $[t, t') \times A$ (with $0 \leqslant t \leqslant t' < \infty$ and $A \subseteq \Sigma$). A timed failure $(s, \aleph)$ is interpreted as an observation of a process in which the events that the process has engaged in are recorded in s, whereas the intervals during which other events have been refused are recorded in $\aleph$. All recorded time values are absolute, i.e., measured relative to the process's 'start'. The set of timed failures is denoted by $TF_{\mathbb{R}}$.

The model $\mathcal{M}_{\mathbb{R}}$ consists of all subsets of $TF_{\mathbb{R}}$ that meet certain 'axioms', general properties that all processes are deemed to have. One then defines the semantic map $\mathbb{R}[\![\cdot]\!] : \mathbf{TCSP} \longrightarrow \mathcal{M}_{\mathbb{R}}$ by induction on the structure of Timed CSP syntax. This map simply assigns to each syntactic process its set of possible behaviours, or timed failures.

Two fundamental assumptions are that processes evolve over a *continuous*, ergo *dense*, time domain ($\mathbb{R}^+$); and that processes are subject to *maximal progress*, or *τ-urgency*, which requires silent events (internal transitions) to occur as soon as they become available. Technical details and more thorough presentations of the timed failures model can be found in any of the references quoted earlier.

A *dense-time specification* is an assertion concerning processes; more precisely, it is a predicate on the set of behaviours of processes. In this paper, the specifications we are interested in are *behavioural*; that is to say, a dense-time specification can always be identified with a set $S \subseteq TF_{\mathbb{R}}$ of timed failures, and a process $P \in \mathbf{TCSP}$ *satisfies* S if $\mathbb{R}[\![P]\!] \subseteq S$; we denote this by $P \vDash_{\mathbb{R}} S$.

As an example, the assertion: "P cannot perform the event *crash*" can be expressed as $P \vDash_{\mathbb{R}} S_1$, where $S_1 = \{(s, \aleph) \in TF_{\mathbb{R}} \mid \mathit{crash} \notin \sigma(s)\}$. (Here $\sigma(s)$ simply denotes the set of events occurring in the timed trace s.)

S_1 is an example of a *(timed) trace* specification: it restricts the set of allowable timed traces of a process, but places no constraints on refusals. Such specifications are often termed *safety* properties, in that they specify that 'nothing bad should happen'. By contrast, a specification restricting the allowable timed refusals of a process is often called a *liveness* property, since it requires that 'certain (presumably good) things not be prevented from happening'. Such a specification, for example, could be the requirement of constant availability of the 'eject' option on a combat aircraft, perhaps expressed as: "The event *eject* is never refused".

A specification S is said to be *syntactic* if it can be written as the set of behaviours of a process, in other words if there exists some $Q \in \mathbf{TCSP}$ such that $S = \mathbb{R}[\![Q]\!]$. In this case we write $P \vDash_{\mathbb{R}} Q$ (rather than $P \vDash_{\mathbb{R}} \mathbb{R}[\![Q]\!]$) to mean $P \vDash_{\mathbb{R}} S$, i.e., $\mathbb{R}[\![P]\!] \subseteq \mathbb{R}[\![Q]\!]$.

It turns out that S_1 above is not syntactic; however, a discrete-time version of S_1 *is* syntactic, as we shall see in the following section. This is an important observation since our model checking algorithm (as well as its implementation on FDR) requires specifications to be expressed syntactically.

As explained earlier, a sort of specification that we are particularly interested in consists in specifications that are *closed under inverse digitisation*, a generalisation of a notion of Henzinger, Manna, and Pnueli [13]. Closure under inverse digitisation can be viewed as a robustness property: if a specification contains all 'digitisations' of a particular behaviour, then it should contain the behaviour in question as well. If we understand specifications to be sets of 'allowable' behaviours, it would seem dangerous to ban a particular behaviour if many neighbouring behaviours were themselves deemed acceptable. We postpone the precise definition of closure under inverse digitisation until Sect. 5.

Given a process P and a specification S, the *dense-time verification* problem we are interested in is deciding whether $P \vDash_{\mathbb{R}} S$ holds. Since P and S are in most cases infinite (in fact uncountable), this is often a far from straightforward task. The remainder of this paper develops techniques to accomplish this goal under certain assumptions.

4 Discrete-Time Modelling, Specification, and Verification

We now present a discrete-time model to interpret Timed CSP processes. As discussed in the introduction, this model, $\mathcal{M}_{\mathbb{Z}}$, is in fact an integral submodel of the timed failures model presented in the previous section.

Processes' behaviours as recorded in $\mathcal{M}_{\mathbb{Z}}$ are precisely those 'integral' behaviours that are recorded in $\mathcal{M}_{\mathbb{R}}$. Moreover, the forthcoming *digitisation lemma* (Lemma 4) tells us that *any* dense-time behaviour of a process recorded in $\mathcal{M}_{\mathbb{R}}$ gives rise to several closely related integral behaviours, so that, modulo some 'timing fuzziness', all of the behaviours of a process that are recorded in $\mathcal{M}_{\mathbb{R}}$ are also accounted for in $\mathcal{M}_{\mathbb{Z}}$.

Integral timed failures are defined in the obvious way: all time values appearing in timed traces, and all time bounds of refusal tokens, are required to be integral. Thus events occur at integral times and are refused over integral intervals of time. We let $TF_{\mathbb{Z}}$ stand for the set of integral timed failures.

The model $\mathcal{M}_{\mathbb{Z}}$ then consists of all subsets of $TF_{\mathbb{Z}}$ that meet certain axioms, essentially integral versions of the axioms for $\mathcal{M}_{\mathbb{R}}$. The semantic map $\mathbb{Z}[\![\cdot]\!] : \mathbf{TCSP} \longrightarrow \mathcal{M}_{\mathbb{Z}}$ can then be defined compositionally in exactly the same way as the dense-time mapping $\mathbb{R}[\![\cdot]\!] : \mathbf{TCSP} \longrightarrow \mathcal{M}_{\mathbb{R}}$, with obvious 'integral' restrictions on the ranges of the various parameters.

For $T \subseteq TF_{\mathbb{R}}$ a set of dense-time timed failures, let $\mathbf{Z}(T) \mathrel{\widehat{=}} T \cap TF_{\mathbb{Z}}$ denote the subset of integral timed failures of T. As expected, we have the following:

Lemma 1. *For any process $P \in \mathbf{TCSP}$, $\mathbb{Z}[\![P]\!] = \mathbf{Z}(\mathbb{R}[\![P]\!])$.*

Proposition 2. *The model $\mathcal{M}_{\mathbb{Z}}$ is strictly coarser than $\mathcal{M}_{\mathbb{R}}$. In other words, there are processes in $\mathbf{TCSP}$ which are identified in $\mathcal{M}_{\mathbb{Z}}$ yet distinguished in $\mathcal{M}_{\mathbb{R}}$ (but never vice-versa).*

Proof. The parenthesised assertion is immediate from Lemma 1. For the main statement, we let $R = \dot{a} \overset{0}{\rhd} \mathit{WAIT}\ 1 \fatsemi R$, and then define $P = R \mid\mid\mid \dot{b}$ and $Q = (R \mid\mid\mid b \longrightarrow R) \underset{\{a\}}{\parallel} \dot{a}$. We now show that $\mathbb{Z}[\![P]\!] = \mathbb{Z}[\![Q]\!]$ but $\mathbb{R}[\![P]\!] \neq \mathbb{R}[\![Q]\!]$.

To see that $\mathbb{Z}[\![P]\!] = \mathbb{Z}[\![Q]\!]$, note that either process may communicate, in any order and at any time, a single a and a single b. Moreover, both processes can refuse everything but b until such time as b occurs, at which point any set of events can be refused.

On the other hand, the timed trace $\langle (0.5, b), (1.5, a) \rangle$ is a valid trace of $\mathbb{R}[\![Q]\!]$ but not of $\mathbb{R}[\![P]\!]$, which concludes the proof. □

Following our approach with $\mathcal{M}_{\mathbb{R}}$, behavioural *discrete-time specifications* on $\mathcal{M}_{\mathbb{Z}}$ processes are naturally (identified with) sets of integral timed failures. Thus given a specification $S \subseteq \mathit{TF}_{\mathbb{Z}}$, we say that a process $P \in \mathbf{TCSP}$ satisfies S, written $P \vDash_{\mathbb{Z}} S$, if $\mathbb{Z}[\![P]\!] \subseteq S$.

We note that dense-time specifications (subsets of $\mathit{TF}_{\mathbb{R}}$) naturally give rise to discrete-time specifications, simply by excluding non-integral behaviours. Thus for $S \subseteq \mathit{TF}_{\mathbb{R}}$, we also write $P \vDash_{\mathbb{Z}} S$ to express $\mathbb{Z}[\![P]\!] \subseteq \mathbf{Z}(S)$.

Let us now consider once again the example "P cannot perform the event *crash*". Understood as a discrete-time specification, it can be written as $P \vDash_{\mathbb{Z}} S_2$, where $S_2 = \{(s, \aleph) \in \mathit{TF}_{\mathbb{Z}} \mid \mathit{crash} \notin \sigma(s)\}$. We could also have expressed S_2 as the set of behaviours of the process $\mathit{CHAOS}_{\Sigma - \{\mathit{crash}\}}$, where we define, for any $A \subseteq \Sigma$, $\mathit{CHAOS}_A = (a : A \longrightarrow \mathit{CHAOS}_A) \overset{0}{\rhd} \mathit{WAIT}\ 1 \fatsemi \mathit{CHAOS}_A$.

A discrete-time specification S is said to be *syntactic* if it can be written as $S = \mathbb{Z}[\![Q]\!]$ for some process Q. S_2 above is therefore syntactic. Note that, thanks to Lemma 1, a dense-time specification that is syntactic is automatically also syntactic when viewed as a discrete-time specification, although the converse need not be true.

Let S be a syntactic discrete-time specification, i.e., $S = \mathbb{Z}[\![Q]\!]$ for some $Q \in \mathbf{TCSP}$. Let $P \in \mathbf{TCSP}$. There is an algorithm to decide whether $P \vDash_{\mathbb{Z}} S$, which is guaranteed to terminate provided the discrete *labelled transition systems* associated with P and Q are finite.[2] In fact, depending on its implementation, the algorithm may also terminate in certain other cases.

The idea underlying the algorithm is to form the *power labelled transition system (PLTS)* of a process, which identifies certain states in such a way that the resulting graph is *deterministic*. One then introduces a notion of *power simulation*, which is subsequently shown to correspond to the satisfaction relation $\vDash_{\mathbb{Z}}$. Since the existence of a power simulation can always be decided in the case of finite PLTS's, this algorithm is a decision procedure for discrete-time specification satisfaction. In fact, the model checker FDR employs similar (if rather more sophisticated) methods for deciding refinement between CSP processes.

[2] A discrete operational semantics, derived from that presented by Schneider for Timed CSP in [25], can be given for **TCSP** processes. This semantics simply associates to each process a discrete labelled transition system, from which the integral timed failures of the process can be calculated—the operational and denotational semantics are said to be *congruent*.

An assertion of the form $P \vDash_{\mathbb{Z}} S$ can also be decided directly on FDR, via simple coding techniques, provided S is a syntactic *trace* specification.[3] We illustrate this in a case study in the following section.

5 Relating Dense-Time and Discrete-Time Verification

We now consider the relationship between the dense-time and discrete-time behaviours $\mathbb{R}[\![P]\!]$ and $\mathbb{Z}[\![P]\!]$ of a given Timed CSP process P.

5.1 The Digitisation Lemma

We first present one of our main results, the *digitisation lemma*, which enables us to tightly relate the dense-time and discrete-time semantics for Timed CSP.

We begin with a small piece of notation. 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: $t = \lfloor t \rfloor + t'$. (Here $\lfloor t \rfloor$ represents the greatest integer less than or equal to t.) If $t' < \varepsilon$, let $[t]_\varepsilon \mathrel{\widehat{=}} \lfloor t \rfloor$, otherwise let $[t]_\varepsilon \mathrel{\widehat{=}} \lceil t \rceil$. (Naturally, $\lceil t \rceil$ denotes the least integer greater than or equal to t.) The $[\cdot]_\varepsilon$ 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 the 'pivot' ε or not.

We can then extend $[\cdot]_\varepsilon$ to timed failures, by pointwise application to, respectively, the time component of the trace's events, and the time bounds of the refusal's tokens:

$$\left[\left(\langle (t_1, a_1), (t_2, a_2), \ldots, (t_k, a_k)\rangle, \bigcup_{i=1}^{l} [u_i, v_i) \times A_i\right)\right]_\varepsilon \mathrel{\widehat{=}}$$
$$\left(\langle ([t_1]_\varepsilon, a_1), ([t_2]_\varepsilon, a_2), \ldots, ([t_k]_\varepsilon, a_k)\rangle, \bigcup_{i=1}^{l} [[u_i]_\varepsilon, [v_i]_\varepsilon) \times A_i\right) \quad .$$

(The reader can verify that, in the case of timed refusals, this operation is independent of the particular representation of the refusal as a finite union of refusal tokens.) This definition extends to sets of timed failures in the obvious way.

Definition 3. *A set of timed failures $P \subseteq TF_{\mathbb{R}}$ is* ***closed under digitisation*** *if, for any $0 \leqslant \varepsilon \leqslant 1$, $[P]_\varepsilon \subseteq P$.*

We note that our version of 'digitisation' extends that presented in [13], which is only concerned with timed traces.

The digitisation lemma now reads:

Lemma 4. *For any $P \in$ **TCSP**, $\mathbb{R}[\![P]\!]$ is closed under digitisation.*

Proof. The proof proceeds by structural induction on Timed CSP terms. Note that the assumption that all delays are integral is crucial. Details of the proof can be found in [18]. □

[3] Certain other types of specifications can also be handled by FDR.

5.2 Verification

Definition 5. *A set of timed failures $\phi \subseteq TF_{\mathbb{R}}$ is* ***closed under inverse digitisation*** *if, whenever a timed failure $(s, \aleph)$ is such that $[(s, \aleph)]_\varepsilon \in \phi$ for all $0 \leqslant \varepsilon \leqslant 1$, then $(s, \aleph) \in \phi$.*

As a convention, we use lowercase Greek letters to designate specifications that are closed under inverse digitisation.

The following lemma extends a result of [13]:

Lemma 6. *If $P \subseteq TF_{\mathbb{R}}$ is closed under digitisation and $\phi \subseteq TF_{\mathbb{R}}$ is closed under inverse digitisation, then $\mathbf{Z}(P) \subseteq \mathbf{Z}(\phi) \Leftrightarrow P \subseteq \phi$.*

On the other hand, suppose $S \subseteq TF_{\mathbb{R}}$ is ***not*** *closed under inverse digitisation. Then there exists $P \subseteq TF_{\mathbb{R}}$ closed under digitisation such that $\mathbf{Z}(P) \subseteq \mathbf{Z}(S)$ yet $P \nsubseteq S$.*

Proof. For the first part, we note that the right-to-left implication is immediate (and requires in fact no assumptions on ϕ).

Going in the other direction, assume that $\mathbf{Z}(P) \subseteq \mathbf{Z}(\phi)$, and let $(s, \aleph) \in P$. Since P is closed under digitisation, $[(s, \aleph)]_\varepsilon \in P$ for all $0 \leqslant \varepsilon \leqslant 1$. Of course, each such digitised behaviour also belongs to $\mathbf{Z}(P)$, and hence to $\mathbf{Z}(\phi)$ by assumption. Since ϕ is closed under inverse digitisation, it follows that $(s, \aleph) \in \phi$, as required.

The second part is straightforward and is left to the reader. □

We now come to the main verification result:

Theorem 7. *Let $P \in$ **TCSP**, and let $S, \phi \subseteq TF_{\mathbb{R}}$ be specifications with ϕ being closed under inverse digitisation. Then*

1. $P \nvDash_{\mathbb{Z}} S \Rightarrow P \nvDash_{\mathbb{R}} S$
2. $P \vDash_{\mathbb{Z}} \phi \Leftrightarrow P \vDash_{\mathbb{R}} \phi$.

Proof. Follows directly from the digitisation lemma and Lemma 6. □

As the above results indicate, the class of specifications that are closed under inverse digitisation is particularly important to us. As pointed out in [13], this class includes *qualitative properties*, as well as *bounded-response* and *bounded-invariance properties*. It is closed under arbitrary intersections (which corresponds to logical conjunction). Lastly, if $P \subseteq TF_{\mathbb{R}}$ is closed under digitisation, then its complement $TF_{\mathbb{R}} - P$ is closed under inverse digitisation. As the reader may easily verify, those facts remain true in the present context.

We also note that, as implied by Lemma 6, specifications that are *not* closed under inverse digitisation cannot directly be verified exactly via digitisation methods; see, however, our discussion on time granularity below.

Unfortunately, closure under inverse digitisation is an undecidable property in Timed CSP, as the next proposition indicates. An interesting problem is to find further straightforward criteria capturing large classes of processes that are closed under inverse digitisation. This task is however made difficult by the fact that closure under inverse digitisation is not preserved by either the parallel or sequential composition operators, nor by (nondeterministic) process refinement.

Proposition 8. *The problem of deciding, given a process $Q \in \mathbf{TCSP}$, whether $\mathbb{R}[\![Q]\!]$ is closed under inverse digitisation is undecidable.*

Proof. This follows immediately from the undecidability of the halting problem for two-counter machines. Counters can be implemented in Timed CSP as processes of the form $C = up \longrightarrow (C \mathbin{|||} down \longrightarrow C)$. If H is a process closed under inverse digitisation and R isn't, deciding whether $Q = H \mathbin{;} R$ itself is closed under inverse digitisation is equivalent to deciding whether H fails to be able to terminate successfully. □

5.3 Time Granularity

Let us define an integral multiplication operation on processes: for $P \in \mathbf{TCSP}$ and $k \in \mathbb{N}^{\geqslant 1}$, define $kP \in \mathbf{TCSP}$ to be identical to P except that every delay n in P has been replaced by a delay of kn in kP. This operation can be given a straightforward inductive definition, which we omit.

We can also define integral multiplication on timed failures, by pointwise application to traces' and refusals' events. This definition, which we also omit, extends to sets of times failures in the obvious way.

For $S \subseteq TF_{\mathbb{R}}$ a dense-time specification, an assertion of the form $P \vDash_{\mathbb{Z}} S$ is only concerned with the integral behaviours of P. One might like to strengthen this requirement to half-integral behaviours of P, i.e., those behaviours which are integral multiples of $1/2$. Indeed, one can strengthen the requirement to $1/k$-th integral behaviours of P for any $k \in \mathbb{N}^{\geqslant 1}$, by considering the assertion $kP \vDash_{\mathbb{Z}} kS$. This, of course, simply corresponds to refining the granularity of time by a factor of k.

In practice, it is often the case that, while S may not be closed under inverse digitisation, $2S$ or kS may be. For example, if $Q = \dot{a} \overset{0}{\rhd} WAIT\ 1 \mathbin{;} Q$ is a process that may communicate a single a at any integral time and then stop, we find that $\mathbb{R}[\![Q]\!]$ is not closed under inverse digitisation, whereas $k\mathbb{R}[\![Q]\!]$ is, for any $k \geqslant 2$.

We are led to the following theorem:

Theorem 9. *Let $P \in \mathbf{TCSP}$, and let $S \subseteq TF_{\mathbb{R}}$ be a dense-time specification. Then, for any $k \in \mathbb{N}^{\geqslant 1}$,*

1. $kP \vDash_{\mathbb{Z}} kS \Rightarrow P \vDash_{\mathbb{Z}} S$
2. $kP \nvDash_{\mathbb{Z}} kS \Rightarrow P \nvDash_{\mathbb{R}} S$
3. *If kS is closed under inverse digitisation, then $kP \vDash_{\mathbb{Z}} kS \Leftrightarrow P \vDash_{\mathbb{R}} S$.*

Proof. This rests on the facts that $\mathbb{Z}[\![kP]\!] \supseteq k\mathbb{Z}[\![P]\!]$ and $\mathbb{R}[\![kP]\!] = k\mathbb{R}[\![P]\!]$. □

Unfortunately, given a specification $S \subseteq TF_{\mathbb{R}}$, it need not be the case that kS be closed under inverse digitisation for *any* value of k; in other words, there are specifications which seemingly cannot be handled (even theoretically) within the sort of discrete frameworks we are considering.

Proposition 10. *There exists a syntactic specification $S \subseteq TF_{\mathbb{R}}$ such that kS is not closed under inverse digitisation, for any $k \in \mathbb{N}^{\geqslant 1}$.*

Proof. Let $Q = \dot{a} \mathbin{|||} a \longrightarrow (\dot{a} \overset{0}{\rhd} STOP)$, and set $S = \mathbb{R}[\![Q]\!]$. Note that $kS = S$ for any $k \in \mathbb{N}^{\geqslant 1}$. Let $s = \langle (0.1, a), (0.2, a), (0.3, a) \rangle$. Observe that $(s, \emptyset) \notin S$, even though every digitisation of $(s, \emptyset)$ clearly belongs to S. □

However, we have the following result:

Theorem 11. *Let $S \subseteq TF_{\mathbb{R}}$ be a syntactic specification, and let $P \in$ **TCSP** be a process. Suppose that $kP \vDash_{\mathbb{Z}} kS$ for arbitrarily large values of k. Then $P \vDash_{\mathbb{R}} S$.*

We remark that the hypothesis that S is syntactic is necessary.

Proof. The proof is somewhat intricate and requires careful analysis. Details can be found in [18]. □

5.4 Full Abstraction

Definition 12. *Let $P, Q \in$ **TCSP** be two processes. We say that P and Q are **equivalent**, written $P \simeq Q$, if for all specifications $\phi \subseteq TF_{\mathbb{R}}$ that are closed under inverse digitisation, $P \vDash_{\mathbb{R}} \phi \Leftrightarrow Q \vDash_{\mathbb{R}} \phi$.*

Our full abstraction result now reads:

Theorem 13. *For any processes $P, Q \in$ **TCSP**, $P \simeq Q \Leftrightarrow \mathbb{Z}[\![P]\!] = \mathbb{Z}[\![Q]\!]$.*

Proof. The right-to-left implication follows directly from Theorem 7.

For the other direction, assume without loss of generality that there exists an integral timed failure $(s, \aleph) \in \mathbb{Z}[\![P]\!]$ such that $(s, \aleph) \notin \mathbb{Z}[\![Q]\!]$. We must show that $P \not\simeq Q$.

Let $\phi \subseteq TF_{\mathbb{R}}$ be the least specification closed under inverse digitisation that contains $\mathbb{Z}[\![Q]\!]$. ϕ can be constructed by adding all the non-integral timed failures to $\mathbb{Z}[\![Q]\!]$ that are required by inverse digitisation. Note that $\mathbf{Z}(\phi) = \mathbb{Z}[\![Q]\!]$. Since $(s, \aleph)$ is an integral behaviour and $(s, \aleph) \notin \mathbb{Z}[\![Q]\!]$, we have $(s, \aleph) \notin \phi$.

Since ϕ is closed under inverse digitisation and $Q \vDash_{\mathbb{Z}} \phi$, Theorem 7 tells us that $Q \vDash_{\mathbb{R}} \phi$. However $P \nvDash_{\mathbb{R}} \phi$, since $(s, \aleph) \in \mathbb{Z}[\![P]\!] \subseteq \mathbb{R}[\![P]\!]$, yet $(s, \aleph) \notin \phi$. Thus $P \not\simeq Q$, as required. □

5.5 Example: Railway Level Crossing

We now present a small verification case study based on a simplified version of the well-known railway level crossing problem [9].[4]

We describe in Timed CSP a closed system made up of four distinct components: trains, travelling at bounded speeds on a stretch of rail incorporating a

[4] See also [23] for an interesting alternative CSP-based discrete-time approach to this example.

level crossing; cars, able to cross the tracks in a bounded amount of time; a traffic light, meant to signal cars not to attempt crossing the railway when a train is nearby; and a monitor, whose rôle is to signal that a collision has happened. For simplicity we assume that only at most one train and one car are present at any one time within the system.

Trains are modelled via the process TRAIN: in its initial state, this process assumes that there are no trains on the tracks, and offers the event $\mathit{train.in}$. This event represents a sensor signal which indicates that an incoming train is at least 60 s away from the crossing. When the train reaches the crossing, the event $\mathit{train.on}$ is triggered, and as soon as the train is a safe distance past the crossing, the event $\mathit{train.out}$ registers. We assume that the train takes at least 20 s to cross the crossing, and that the process TRAIN returns to its initial state as soon as the event $\mathit{train.out}$ is received.

The process CAR models the cars: initially there are no cars on the crossing, and CAR offers the event $\mathit{car.on}$, subject to synchronisation with the traffic light, indicating that a car is just about to drive onto the crossing. The car stays in this vulnerable position for at most 10 s, sending out the event $\mathit{car.out}$ as soon as it is safely on the other side. For simplicity we will make the conservative assumption that the time taken to cross the tracks is actually exactly 10 s. In order to ensure that the car does step out immediately after this time, however, we will later on hide the event $\mathit{car.out}$, to enforce its urgency. A new car is allowed on the crossing as soon as the car ahead has left it.

The traffic light is green to start with, modelled by the process GREEN, and becomes red as soon as a train ($\mathit{train.in}$) is detected. While it is red, the event $\mathit{car.on}$ is disabled (modelling the assumption that any car not yet on the crossing obeys the red light), and is only re-enabled after $\mathit{train.out}$ has registered.

A collision will occur if the train enters the crossing while a car is already there, or vice-versa; in either case this will cause the monitoring process WATCH to send out the catastrophic event crash.

The entire level crossing system $\mathit{CROSSING}$ is modelled as the parallel composition of these four components, with $\mathit{car.out}$ hidden.

Translating this description into Timed CSP, we get:

$$\begin{aligned}
\mathit{TRAIN} &= \mathit{train.in} \xrightarrow{60} \mathit{train.on} \xrightarrow{20} \mathit{train.out} \longrightarrow \mathit{TRAIN} \\
\mathit{CAR} &= \mathit{car.on} \xrightarrow{10} \mathit{car.out} \longrightarrow \mathit{CAR} \\
\mathit{GREEN} &= (\mathit{train.in} \longrightarrow \mathit{RED}) \,\Box\, (\mathit{car.on} \longrightarrow \mathit{GREEN}) \\
\mathit{RED} &= \mathit{train.out} \longrightarrow \mathit{GREEN} \\
\mathit{WATCH} &= (\mathit{train.on} \longrightarrow \mathit{WATCH}_{\mathit{train}}) \,\Box\, (\mathit{car.on} \longrightarrow \mathit{WATCH}_{\mathit{car}}) \\
\mathit{WATCH}_{\mathit{train}} &= (\mathit{car.on} \longrightarrow \mathit{crash} \longrightarrow \mathit{STOP}) \,\Box\, (\mathit{train.out} \longrightarrow \mathit{WATCH}) \\
\mathit{WATCH}_{\mathit{car}} &= (\mathit{train.on} \longrightarrow \mathit{crash} \longrightarrow \mathit{STOP}) \,\Box\, (\mathit{car.out} \longrightarrow \mathit{WATCH}) \\
\mathit{CROSSING} &= ((\mathit{TRAIN} \underset{A}{\parallel} (\mathit{GREEN} \underset{B}{\parallel} \mathit{CAR})) \underset{C}{\parallel} \mathit{WATCH}) \setminus \{\mathit{car.out}\}
\end{aligned}$$

where $A = \{\mathit{train.in}, \mathit{train.out}\}$, $B = \{\mathit{car.on}\}$, and $C = \{\mathit{train.on}, \mathit{car.on}, \mathit{train.out}, \mathit{car.out}\}$.

We would now like to prove that this system is 'safe', i.e., that no collision can ever occur. To this end, let us define the specification SAFE to be: 'The event *crash* is never witnessed'; in other words, $\mathit{SAFE} = \{(s, \aleph) \in \mathit{TF}_{\mathbb{R}} \mid \mathit{crash} \notin \sigma(s)\}$. We aim to establish that $\mathit{CROSSING} \vDash_{\mathbb{R}} \mathit{SAFE}$.

SAFE is a qualitative behavioural timed trace specification, and is therefore closed under inverse digitisation (as can also be seen by inspection). It follows from Theorem 7 that $\mathit{CROSSING} \vDash_{\mathbb{Z}} \mathit{SAFE} \Leftrightarrow \mathit{CROSSING} \vDash_{\mathbb{R}} \mathit{SAFE}$.

For model checking purposes, recall that, as claimed in Sect. 4, $\mathbf{Z}(\mathit{SAFE}) = \mathbb{Z}[\![\mathit{CHAOS}_{\Sigma - \{\mathit{crash}\}}]\!]$, where $\Sigma = \{\mathit{train.in}, \mathit{train.on}, \mathit{train.out}, \mathit{car.on}, \mathit{car.out}, \mathit{crash}\}$ represents $\mathit{CROSSING}$'s alphabet. The question therefore reduces to deciding whether $\mathit{CROSSING} \vDash_{\mathbb{Z}} \mathit{CHAOS}_{\Sigma - \{\mathit{crash}\}}$ holds.

This can be encoded and checked as a trace refinement using the model checker FDR. A special event *tock* is introduced to represent the passage of one time unit. Most of the CSP operators must be modified to consistently handle the passage of time; this is achieved in most cases via simple coding tricks. The most significant difficulty is enforcing τ-urgency, the requirement that time cannot pass while hidden events are on offer. This is taken care of by invoking the *priority* operator of FDR which simply disables *tock*-transitions whenever τ-transitions are pending.

Since both $\mathit{CROSSING}$ and $\mathit{CHAOS}_{\Sigma - \{\mathit{crash}\}}$ are finite state (as they only comprise *tail recursions*), the FDR check is guaranteed to terminate. Indeed, it did confirm that $\mathit{CROSSING} \vDash_{\mathbb{R}} \mathit{SAFE}$, as surmised.

6 Conclusion

The main results of this paper are that Timed CSP processes are closed under digitisation, and as a consequence that specifications closed under inverse digitisation can be verified exactly using an integral behaviour model. This yields a model checking algorithm, implementable on FDR, which however requires that specifications be expressed as syntactic processes. As our case study suggests, this should not turn out to be a serious impediment in practice. Another important result is that integral behaviours are fully abstract with respect to specifications closed under inverse digitisation, and therefore that checks performed by our model checking algorithm are not only sufficient, but also necessary, in general, for verification purposes.

For reasons of space, we have presented our work exclusively within a denotational context. As explained in the introduction, our results apply equally well to the operational setting (cf. [18]). Our restriction to *behavioural* specifications, on the other hand, is arguably much more difficult to lift. We do note, however, that many important non-behavioural specifications, such as *reachability* requirements, can perfectly well be handled within our framework.

Our focus on Timed CSP was convenient, but not necessary: the results presented here are reasonably robust and should carry over without great difficulty to other settings, whether process algebraic or transition-systems-based. We refer the reader to [18] for a more detailed discussion on the matter.

The question of the efficiency of the algorithm presented here has only been briefly discussed. As with every other approach to the subject of automated verification, trade-offs are inevitable. We expect our algorithm to prove reasonably efficient in cases where all delays are relatively small, or at least of similar sizes. This is a topic for further research.

The question of full abstraction is interesting and does not seem to have been extensively studied, whether in relation to algorithmic efficiency or not. The matter resurfaces as soon as we vary the basic parameters under consideration (the process algebra, the verification framework, the class of allowable specifications). For instance, are region graphs fully abstract with respect to TCTL formulas? This, too, seems an interesting starting point for further work.

Acknowledgements

I am grateful to Mike Reed, Bill Roscoe, Steve Schneider, James Worrell, Mike Mislove, Michael Goldsmith, and Formal Systems for fruitful discussions and support with FDR. Thanks are also due to the anonymous referees for their careful review and insightful suggestions.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS 90)*, pages 414–425. IEEE Computer Society Press, 1990.

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

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

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

[5] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[6] R. Alur and R. P. Kurshan. Timing analysis in Cospan. In *Proceedings of Hybrid Systems III*, volume 1066, pages 220–231. Springer LNCS, 1996.

[7] J. Bengtsson, K. G. Larsen, F. Larsen, P. Pettersson, and W. Yi. UppAal: A tool-suite for automatic verification of real-time systems. In *Proceedings of Hybrid Systems III*, volume 1066, pages 232–243. Springer LNCS, 1996.

[8] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proceedings of Hybrid Systems III*, volume 1066, pages 208–219. Springer LNCS, 1996.

[9] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proceedings of the Tenth International Workshop on Real-Time Operating Systems and Software*, 1993.

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. In *Proceedings of the Ninth International Conference on Computer-Aided Verification (CAV 97)*, volume 1254, pages 460–463. Springer LNCS, 1997.

[11] T. A. Henzinger and O. Kupferman. From quantity to quality. In *Proceedings of the First International Workshop on Hybrid and Real-time Systems (HART 97)*, volume 1201, pages 48–62. Springer LNCS, 1997.

[12] T. A. Henzinger, O. Kupferman, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proceedings of the Seventh International Conference on Concurrency Theory (CONCUR 96)*, volume 1119, pages 514–529. Springer LNCS, 1996.

[13] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of the Nineteenth International Colloquium on Automata, Languages, and Programming (ICALP 92)*, volume 623, pages 545–558. Springer LNCS, 1992.

[14] 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.

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

[16] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proceedings of the Sixth International Conference on Concurrency Theory (CONCUR 95)*, volume 962, pages 27–41. Springer LNCS, 1995.

[17] J. Ouaknine. Specification as refinement in timed systems. In preparation.

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

[19] J. Ouaknine and G. M. Reed. Model-checking temporal behaviour in CSP. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 99).* CSREA Press, 1999.

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

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

[22] G. M. Reed, A. W. Roscoe, and S. A. Schneider. CSP and timewise refinement. In *Proceedings of the Fourth BCS-FACS Refinement Workshop*, Cambridge, 1991. Springer WIC.

[23] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall International, London, 1997.

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

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

[26] S. A. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28:43–90, 1997.

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

[28] O. V. Sokolsky and S. A. Smolka. Local model checking for real time systems. In *Proceedings of the Seventh International Conference on Computer-Aided Verification (CAV 95)*, volume 939, pages 211–224. Springer LNCS, 1995.

[29] Y. Yu, P. Manolios, and L. Lamport. Model checking TLA$^+$ specifications. In *Proceedings of the Tenth Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 99)*, volume 1703, pages 54–66. Springer LNCS, 1999.