DOI 10.1007/s00165-012-0251-6
BCS © 2012
Formal Aspects of Computing (2013) 25: 3–35

Formal Aspects of Computing

Checking noninterference in Timed CSP

A. W. Roscoe and Jian Huang
Department of Computer Science, Oxford University, Parks Road, Oxford OX1 3QD, UK

Abstract. A well-established specification of noninterference in CSP is that, when high-level events are appropriately abstracted, the remaining low-level view is deterministic. This is not a workable definition in Timed CSP, where many processes cannot be refined to deterministic ones. We argue that in fact "deterministic" should be replaced by "maximally refined" in the definition above. We show how to automate the resulting timed noninterference check within the context of the recent extension of FDR to analyse a discrete version of Timed CSP, and how an extended theory of digitisation has the potential both to create more accurate specifications and to infer when processes are noninterfering in the more usual continuous-time semantics.

Keywords: CSP, Timed CSP, Noninterference, Refinement

1. Introduction

Noninterference, a concept introduced by Goguen and Meseguer in [GoMe82], is a topic in the theory of computer security: it analyses whether information can flow between users of a system through their joint use of it. In the classic set up there is a high level user (say Hugh) and a low level one (Lois): we might well want to ask whether or not information can pass from Hugh to Lois. Thus noninterference is an asymmetric condition: we might not mind Hugh learning about Lois's activities.

In a practical setting it might be much harder to guard against situations where Hugh is actively trying to pass information to Lois using whatever feature of the system he can (something usually called a *covert channel*), as opposed to Lois spying on an unknowing Hugh. However, without knowing exactly what Hugh might do, proving the absence of information flow in the second scenario is the same as the in the first.

Noninterference is a wonderful specification for theorists to play with, because it exercises the nuances of their semantic models—something that will be well illustrated in the present paper. Given that semantic models tend to be based on things that observers at some level of abstraction can see about processes, it seems natural to pose the question of how one would couch noninterference by giving Lois the same powers of observation as the model, though restricted to her own interface with the system. So while refining a semantic model may be irrelevant to many practical specifications, this is rarely if ever true of noninterference because we can always imagine a more discerning Lois or spy.

Correspondence and offprint requests to: A. W. Roscoe, E-mail: Bill.Roscoe@cs.ox.ac.uk

Goguen and Meseguer's specification was in terms of machines that strictly alternate inputs and outputs, and on each cycle have an input, then an output, with each of its users. Process algebras like CSP offer a rather more flexible way of describing how processes look, and since they are essentially ways of describing *interaction*, quickly became an important focus of noninterference research. Initial characterisations in process algebras (for example [All91, GCu92, Rya91]) had much in common with those of [GoMe82], but later ones such as those of Roscoe, Woodcock and Wulf [RWW96, Ros95], and Focardi and Gorrieri [FoGo94] made much more use of the particular expressive qualities of process algebras. It is in this world that the present paper sits.

Different types of semantics (whether process algebra or otherwise) give different perspectives on noninterference. If the semantic model we are using does not capture some notion of behaviour that Lois might observe, then any specification of noninterference based on that model is not going to capture information about Hugh's actions that she can see in that way. One obvious possibility is time. Neither standard input/output semantics of sequential programs nor most process algebras pay any attention to how long our system takes to perform its operations, or the wait between one communication and the next. Therefore none of the formulations of noninterference in the papers cited above can identify *timing channels*, one of the most common types of covert channel. The formulations we give in this paper are designed for exactly that purpose.

A further important question is whether or not we try to distinguish different sorts of nondeterminism—unpredictable behaviour by the system—from one another. Nondeterminism can either protect against information flow or allow it. If what Lois sees covers the same nondeterministic range no matter what Hugh does it is impossible for her to deduce anything definitively about his actions, but on the other hand what he does (for example his timing) might affect the resolution of the nondeterminism in her view.

The relationship between refinement, nondeterminism and noninterference has generated considerable debate over the years. In particular one needs to be careful not to describe a system as secure and yet find it has insecure refinements—an instance of the so-called *refinement paradox*. Morgan and McIver have written about this issue—mainly in the context of imperative programs [Mor06, McIMo10], for example, by introducing shadow variables that assert unrefinable ignorance. We will find in this paper that there seems to be a greater need in Timed CSP than in the original CSP for ways of determining whether or not nondeterministic programs, possibly not maximal in the conventional refinement order, satisfy noninterference.

In this paper we will show that the determinism-based specification of noninterference has to be altered subtly in the world of Timed CSP. But before we can demonstrate that, we need to build quite a lot of background knowledge.

The rest of this paper is therefore structured as follows. We first recall the CSP and Timed CSP languages. We then introduce and analyse the continuous and discrete semantic models we use for Timed CSP, deriving some new structural properties. A further background section recalls the functionality of FDR including its new timed capabilities. Section 5 recalls the definition of noninterference from [RWW96, Ros95], and shows that it does not work in the same form in the standard semantics for Timed CSP (which uses a continuous model of time) or the corresponding model using discrete time. We then show how a revised formulation of the basic principles allows us to capture what is required.

The theory of *digitisation* [Oua01, Oua02] allows us to relate the behaviour of Timed CSP processes in discrete and continuous time. We investigate the implications of this for systems modelled in the continuous models of Timed CSP, and show that one can provide results about the noninterference properties of a process's continuous semantics by analysing a suitable discrete semantics. To do this we establish some generalised results about digitisation.

The discrete-time form of Timed CSP has recently [ALOR12] been implemented in the CSP refinement checker FDR [Ros94]. We show that the reformulated definition of noninterference can be implemented directly in that, though not so directly as the untimed variant, since it cannot use FDR's built-in determinism check. By applying this to some relatively simple case studies, we see some of the types of timing channels that can arise in shared-use systems and some strategies for avoiding these.

There is a great deal of literature on noninterference in untimed contexts, but less in timed ones. In Sect. 9 we compare our work with what seem to be the two most similar: the presentation in [FGM00, FGM03] of noninterference in the context of a *tock*-CSP like process algebra called tSPA, and formulations of noninterference for timed automata in [BFST02, BaTe03]. Perhaps surprisingly, the second of these turns out to be closer, technically, to ours.

Finally, we contemplate potential application areas of timed noninterference analysis, including Cloud security.

The reader will discover that some of the constructions and arguments contained in this paper are complex. To keep them as simple as possible we make a number of assumptions:

1. The alphabet Σ over which we build processes is finite.
2. We only consider finitely nondeterministic CSP and Timed CSP: given the first assumption this just means that all uses of nondeterministic choice $\sqcap$ are over finite sets.
3. While we frequently use termination ($\checkmark$) and sequential composition in building process descriptions, the processes we test for noninterference will never terminate. We therefore ignore the complications of termination when defining and analysing the semantic models and in formulating noninterference conditions.

The present paper has its origins in the doctoral research of the second author [Hua10], some of which relating to discrete Timed CSP was reported in [HuRo06]. His thesis examined a variety of timed models of CSP-like processes and explored how established noninterference theories extended to them. We are now able to extend the parts of [Hua10] on discrete and continuous Timed CSP by proving relationships between them, and implementing decision procedures for them in FDR.

2. The language of CSP and Timed CSP

CSP is a language which describes patterns of communication in some alphabet Σ of actions that are handshaken between the process and its environment. There are additional actions $\checkmark$, a signal for successful termination and τ, an invisible action representing internal progress within a process. In the original "untimed" treatment of CSP these patterns of communication include the order of actions, which sets are offered and maybe the ways in which possibilities branch, but not the exact times these things happen.

The following is a brief introduction to the main parts of the language: much more complete explanations can be found elsewhere [Ros97, Sch00, Ros10].

There are process constants representing important patterns of communication: $STOP$ is a process that does nothing, while $SKIP$ just terminates. **div** just *diverges* by performing τs for ever. RUN_A is always ready to perform any event from $A \subseteq \Sigma$ while $CHAOS_A$ can always both accept and refuse any event from A.

There are operators for introducing communications: $a \rightarrow P$ and $?x : A \rightarrow P(x)$ allow an individual member or choice of actions from Σ. $P \Box Q$ makes the choices of both P and Q available to the environment, while $P \sqcap Q$ allows the process to select which of P and Q to behave like.

We can put processes in parallel that influence each other by synchronising on some of their events from Σ: $P \underset{A}{\parallel} Q$ makes them synchronise on the events they perform in A, $P \ {}_A\|_B \ Q$ makes them synchronise in $A \cap B$, while $P \,|||\, Q$ just lets them run freely.

$P \setminus A$ represents P running but with all events in A hidden: turned into τs. $P[\![R]\!]$ applies the relation $R \subseteq \Sigma \times \Sigma$ (usually assumed to be total on the events P uses) to P's actions: whenever P performs a, $P[\![R]\!]$ gives the environment the choice of all the b such that $a \, R \, b$.

CSP provides three ways of one process handing over to another P; Q runs them in sequence: P until it terminates via $\checkmark$ and then Q. $P \triangle Q$ allows Q to *interrupt* P by performing any visible event, while $P \, \Theta_A \, Q$ runs P until it performs any action in $A \subseteq \Sigma$, at which point Q starts.

There are also indexed versions of a number of these operators, and in many contexts we use infinitary versions of the choice operators. Recursive definitions are used in a rich variety of ways, including defining infinite families of processes in mutual recursions. CSP is therefore a very rich language and is capable of describing many patterns representing both implementations and specifications.

In this paper we will use the "blackboard" style of the operators used above, but in fact our implementations of the ideas in this paper are all in the ASCII version of CSP, known as CSP_M which combines ASCII versions of the above operators with a Haskell-like functional programming language.

Timed CSP [Ree88, ReRo88, Sch00] does not need much more description because it is the same language given a timed interpretation. In our treatment there is only one new construct: $WAIT\,t$ behaves exactly like $SKIP$ but takes the non-negative time t before it terminates ($\checkmark$). As implemented in FDR it gives the programmer the option to assign a non-negative completion time to each event $a \in \Sigma$: in $a \rightarrow P$ and $?x : A \rightarrow P(x)$ there are $et(a)$ time units between the occurrence of the event a (assumed to be instantaneous) and the following process starting up. The other main principle underlying the timing is that we assume that as soon as any event is enabled in a process (either because like τ it needs no collaboration from the environment, or because the environment does allow it) some event does happen. This is the principle of *maximal progress*.

Some versions of Timed CSP include explicit time-out and timed interrupt operators: $P \rhd_t Q$ offers initial choices of P for time t and then lets Q take over if P has not communicated; $P \triangle_t Q$ makes Q take over after time t even if P has communicated (unless P has terminated). But since both of these can be defined in terms of $WAIT$ and other operators, we will not regard these as primitive operators here.

- $P \rhd_t Q = (P \,\Box\, (WAIT\,t;\ timeout \rightarrow Q)) \setminus \{timeout\}$ where $timeout$ is a new event with $et(timeout) = 0$.
- $P \triangle_t Q = (P \,\triangle\, (WAIT\,t;\ interrupt \rightarrow Q)) \setminus \{interrupt\}$ where $interrupt$ is a new event with $et(interrupt) = 0$.

Note how both these constructions depend on maximal progress: as soon as the fresh event becomes available it happens, forcing either resolution of the $\Box$ or an interrupt.

The natural timed interpretations of the processes **div**, RUN_A and $CHAOS_A$ all breach an important principle of Timed CSP: they allow an infinite number of events in a finite time. Timed CSP assumes to the contrary that processes only perform a finite number in any finite interval. What we assume in this paper is any complete Timed CSP system has this property: it can sometimes be useful to use RUN_A and $CHAOS_A$ in these programs as long as they synchronise all their actions with a process that does have the *no-Zeno* property.

We will find in this paper that it is frequently useful to restrict Timed CSP so that all delays introduced by language constructs are integers. In other words, $et(a) \in \mathbb{N}$ and, in $WAIT\,t$, $t \in \mathbb{N}$. The language subset satisfying this restriction will be termed *integer Timed CSP*.

3. Semantic models

CSP, and similarly Timed CSP, are distinguished by the fundamental role that abstract *behavioural* models play in their semantics. These models, sets of observations that can be made of the process under consideration, must both be congruences (i.e. you can calculate the value of any operator applied to its arguments from the values of the arguments) and each have a theory that allows us to construct a fixed-point semantics for recursion. These requirements tell us that there is a *denotational* semantics for the language over it. Each such model yields a theory of *refinement* that is defined by $P \sqsubseteq Q$ if and only if $P = P \sqcap Q$ or equivalently $P \sqsupseteq Q$: Q refines P if Q has less observable behaviours than P.

Refinement is generally identified, conceptually, with reduction in nondeterminism. The richer the language of behaviours captured by a given model, the better this analogy is. (Timed) CSP contexts are always monotonic with respect to refinement: $P \sqsubseteq Q \Rightarrow C[P] \sqsubseteq C[Q]$.

There are invariably two quite distinct ways of calculating the semantics of any process P defined in (Timed) CSP in such a model $\mathcal{M}$. One can either do so via the denotational semantics entirely within $\mathcal{M}$, or one can take the operational semantics of P and formally observe the relevant sets of behaviours. It is always a requirement that these two views co-incide: this is what it means for a semantics to be operationally congruent, and all the semantics discussed in this paper have that quality. Full operational[1] and denotational semantics for Timed CSP can be found in [Sch00].

[1] The operational semantics of Timed CSP [Sch00] has three types of action: the usual visible and τ actions, and timed evolution $P \overset{t}{\rightsquigarrow} Q$. Maximal progress means that a state with a τ action cannot also have a non-zero timed evolution. Timed evolution is dense and deterministic, in the sense that if $P \overset{t_1+t_2}{\rightsquigarrow} R$ then there is Q such that $P \overset{t_1}{\rightsquigarrow} Q$ and $Q \overset{t_2}{\rightsquigarrow} R$; and $P \overset{t}{\rightsquigarrow} Q \wedge P \overset{t}{\rightsquigarrow} Q'$ implies $Q = Q'$. They have the property that if $P \xrightarrow{a} R$ and $P \overset{t}{\rightsquigarrow} Q$ then there exists R' with $Q \xrightarrow{a} R'$. Thus any set of events that is offered continues to be offered until some action (visible or τ) occurs.

Semantic models are given *healthiness conditions*, which characterise which sets of behaviours can reasonably be reckoned to be descriptions of processes. They generally, in fact, capture which[2] sets of behaviours are images the operational semantics. The semantic model is defined to be the collection of sets of the relevant sort(s) of behaviour that satisfy the healthiness conditions.

The most standard model of untimed CSP is the failures-divergences model $\mathcal{N}$ alluded to above in which each process is represented as a pair (F, D) of sets of behaviours. F comprises *failures*, namely combinations (s, X) of a finite trace s and a set X that the process can refuse in a stable state after s. D is the set of traces on which it can *diverge*, namely engage in an infinite consecutive sequence of τ actions. The model is *divergence strict*, namely if $s \in D$ then $s^\frown t \in D$ and $(s^\frown t, X) \in F$ for all t and X: this does not imply that the process can really perform all these extra behaviours, but rather than we choose not to know whether it can or not and simply *assume* they are.

Aside from these divergence-closure conditions, the observation that if (s, X) is a failure then so is (s, Y) for $Y \subseteq X$ and the property that the set of all traces $\{s \mid (s, X) \in F\}$ is nonempty and closed under prefix, there is one further healthiness condition that characterises which pairs (F, D) represent realistic processes. That is

F3 $\quad (s, X) \in F \wedge Y \subseteq \{a \mid (s^\frown\langle a\rangle, \emptyset) \notin F\} \Rightarrow (s, X \cup Y) \in F$

In other words, whenever our process refuses a set X it must also refuse (if offered) an extension of X by events that are *never* possible after s. **F3** ensures that the process has enough traces to be consistent with its refusal sets.

With exception of additional properties that are used to govern the behaviour of the special event $\checkmark$ representing successful termination (which we ignore, for simplicity, in this paper), these properties completely determine $\mathcal{N}$ over any given alphabet Σ. This model has many important properties, one of which is that (given the CSP language described in [Ros10]) *fully abstract* with respect to deciding whether any process is *deterministic* in the following sense:

- A deterministic process (F, D) is divergence free (i.e. $D = \emptyset$)
- It never has the choice whether to accept or refuse any event (i.e. if $(s^\frown\langle a\rangle, \emptyset) \in F$ then $(s, \{a\}) \notin F$), which is equivalent to saying that the failures (s, X) are exactly the ones forced from the set of traces by the property **F3** above.

This concept of determinism is one that relates to how a process can be observed rather than its internal construction. For example the operational semantics of the process $((a \rightarrow b \rightarrow a \rightarrow STOP) \sqcap b \rightarrow STOP) \setminus \{a\}$ branch, so one cannot be certain what state one is in after any trace, but no matter what happens a b followed eventually by $STOP$ occurs.

As you would expect from our discussion earlier, the deterministic processes are precisely the refinement-maximal processes in $\mathcal{N}$.

A wide range of other models of untimed CSP are described in [Ros10], but $\mathcal{N}$ is the most important with respect to specifying noninterference.

The combination of the principle of maximal progress and the need to make models compositional under the CSP hiding operator (which turns visible actions into τs that are forced before time passes $P \setminus X$ can only let time pass when P refuses the whole of X, so it follows that whatever model we are using for P must give us this information) makes the range of models for Timed CSP more restricted than for untimed. It is necessary to record the set of events refused at every point in a behaviour where time advances: time can only advance in $P \setminus X$ in states where the whole of X is refused by P. Divergence is a much reduced issue, since thanks to the no-Zeno assumption any divergence is necessarily spread over infinite time—which when we are modelling time simplifies things greatly. In fact divergence will not be considered in the models we use in this paper.

[2] As shown in [Ros10], for example, this characterisation tends to be exact in the world of untimed CSP provided one allows oneself to use complex infinite mutual recursions and sometimes infinite nondeterminism. We are not aware of similar results for Timed CSP. We suspect that one route to proving them will be the quasi-deterministic processes discussed later in this paper.

In the case of continuous time this means that we have to record refusals as a subset of $\Sigma \times \mathbb{R}^+$ to accompany *timed traces* which attach a time in $\mathbb{R}^+$ (the non-negative real numbers) to each event, where the times increase, not necessarily strictly, through the trace. In fact, timed refusals are unions of sets of the form $X \times [t_1, t_2)$ where $0 \leq t_1 < t_2 < \infty$—*refusal tokens*. $[t_1, t_2)$ is a *half-open interval* that contains t_1, all x with $t_1 \leq x < t_2$ but not t_2. This corresponds to the idea that if an event happens *at* time t then the refusal recorded at that time is the set of events refused at the same time *after* the event. So in $a \to P$, there will be behaviours in which a occurs at time 1, all events other than a are refused in the interval $[0, 1)$ and, on the assumption that the event a takes time δ to complete, all events including a are refused in the interval $[1, 1 + \delta)$.

So the *Timed Failures model* consists of sets of pairs of the form $(t, \aleph)$, where t is such a timed trace,[3] and $\aleph$ is such a timed refusal. First introduced in [ReRo88], there have been a number of variants of this model over the years, the main points of difference being:

(1) Is causality permitted between simultaneous events: can one have the timed trace $\langle (a, 1), (b, 1)\rangle$ but not the timed trace $\langle (b, 1), (a, 1)\rangle$?

(2) Can an event take zero time: in $a \to b \to P$ can the b happen at the same time as the a? (This question is very closely linked to the previous one: if the answer to this one is "yes", then the first must also be answered affirmatively.)

(3) Does recursion take time to unfold or not: is $\mu\, p.F(p)$ equivalent to $WAIT\,\delta;\ F(\mu\, p.F(p))$ for some $\delta > 0$ or just $F(\mu\, p.F(p))$?

(4) How is the assumption of no-Zeno behaviour enforced? This says that only finitely many actions can occur in a finite time.

(5) Is information included about *stability*? This is the dual of divergence: the time after which no further internal actions occur without a visible one having occurred first.

(6) Can traces s and timed refusals $\aleph$ extend through all time or must they be finite? In either case they are always restricted so that up to any finite time they only have finitely many events or are the union of finitely many refusal tokens.

[Oua01, Oua02] and [Sch00] agree on most of these points, and give the same equivalence over Timed CSP restricted to finitely nondeterministic constructs. We agree with them on all of them except the one where they differ, which is the last. So the answers to all but that question will be (1) yes; (2) yes; (3) no; (4) usually restricting recursions to ones which are *time guarded*, never starting a recursive call until some delay of at least $\delta > 0$ has been introduced by the context of the call; and (5) no.

On point (6) we will restrict to finite traces but allow timed refusals extending through all time. They will, however, be restricted so that only finitely many refusal tokens $A \times [t_1, t_2)$ in the union yielding an $\aleph$ have $t_1 < T$ for any fixed T. Both these decisions make later constructions easier, but under our assumption of finite nondeterminism do not change the expressive power of the model: the infinite timed refusal $(s, \aleph)$ (for finite timed trace s) belongs to a process if and only if $(s, \aleph \restriction t)$ for every $t > 0$. (The $\Rightarrow$ direction of this does not depend on finite nondeterminism.)

The axiom that coincides with **F3** is more complex both because it deals with the richer structure of timed failures and because it captures the temporal concept of *no instantaneous withdrawal* or **NIW**: if a process cannot refuse an event up to a given time, then it can perform it at that time. The intuition here is that if event a was offered before time t when some internal event x occurred that removed the option of a, then a was a valid option to x at the point where it occurred and so could have happened instead. This property is important to the theory of digitisation and will play a major role in this paper. In this paper we will call it the continuous forcing axiom **CF**.

$$(s, \aleph) \in P \Rightarrow \exists\, \aleph' \supseteq \aleph.\, \forall\, t.\, \forall\, a.$$
$$(\neg\, \exists \epsilon > 0.\{a\} \times [max\{end(s \restriction t), t - \epsilon\}, t + \epsilon) \subseteq \aleph') \Rightarrow (s \restriction t^\frown\langle (a, t)\rangle, \aleph' \restriction t) \in P$$

This says that whatever timed refusal $\aleph$ is actually observed by an experimenter alongside the timed trace s, there is an extension $\aleph'$ which records all the events the process would have refused (if offered) alongside the trace s through all time. The fact that these are *all* the refused events means that it must perform all actions that are either not in the set *or have been withdrawn from the offer at the present instant*.

[3] There is a summary of some of the notation of timed traces and timed refusals at the end of this paper.

This is intuitive except for the italicised last clause, which says that actions are still possible at the moment of withdrawal. We can give this the following operational explanation. Suppose our process is in operational state P giving rise to a refusal. Suppose it has the set of actions A immediately available. A cannot include τ, which would force something to happen immediately so no refusal would be apparent in P. Then the process will not be refusing anything in A, and will not do so until state P is left. The only way in which the process can start refusing $a \in A$ without a visible action happening is if a τ becomes enabled in P at some later time t and occurs, leading (either directly or after further τs at the same time) to a state P' where both τ and a are impossible. Here τ can happen when it becomes enabled at t. But the whole of A, including a, are available as alternatives to τ at the exact time t. So the process can both perform a at time t and (through the alternative behaviour) refuse it after the same timed failure has been observed through $0, t$).

The **NIW** property comes about because reaching a particular time in the operational semantics *enables* new actions in addition to existing ones, rather than representing a complete change in state. It is more a consequence of the assumed operational model than the language of Timed CSP. If we had a language or operational model in which reaching a time can (of itself) disable previously available actions (and we will see such a language, though not for continuous time, in Sect. 9) then we would use a version of **CF** without **NIW**. This would be the same except that the interval $[max\{end(s \upharpoonright t), t-\epsilon\}, t+\epsilon)$ would be simplified to $[t, t+\epsilon)$. Many of the more interesting (or difficult, depending on your point of view) features of the rest of this paper would then not apply. We will gain more insight into this issue in Sect. 9.

The only other properties required to define the timed failures model are the following

NE A process is non-empty, specifically containing $(\langle\rangle, \emptyset)$.

IC Whenever $(s^\frown s', \aleph) \in P$ and $\aleph' \subseteq \aleph \upharpoonright begin(s')$, then $(s', \aleph') \in P$, including the case where $s' = \langle\rangle$ and so $begin(s') = \infty$. The process is thus *implication closed*: if the presence of timed failure $(s, \aleph)$ implies the presence of $(s', \aleph')$ then the set of behaviours representing it always respects this. (This corresponds to both the prefix-closure of traces and subset-closure property of refusals in the untimed failures models.)

NZ For each P there is a bound on how many events it can perform in any finite time:

$$\forall t. \exists n. s \in traces(P) \Rightarrow \#s \upharpoonright t \leq n$$

Our assumptions of finite alphabet and finite nondeterminism make this last property unproblematic. This is a version of the *no-Zeno* assumption.

The (continuous) timed failures model $\mathcal{F}_{\mathbf{T}}$ will be the set of all sets P of timed failures satisfying all of the above.

A corresponding model $\mathcal{F}_{\mathbf{DT}}$ exists for discrete time: where time is measured in discrete units, separated by some marker such as an event *tock* representing the regular passage of time.[4] $\mathcal{F}_{\mathbf{DT}}$ and variants have been studied and described in [Oua01, Oua02, OuWo03, LoOu06, Hua10, Ros10]. In Timed CSP the processes do not communicate this event: you should think of it as a clock in the hands of an external observer. The *discrete timed failures model* has behaviours that consist of a trace consisting of events including *tock*, including a refusal set of events before each *tock* and at the end. None of the refusal sets include *tock*. Another equivalent presentation (which we will not be using in this paper, but can help to explain the model's structure) is as a sequence of failures (s, X) where there is a notional *tock* between each consecutive pair. For consistency with the continuous treatment above, in this paper we will assume that the behaviour is infinite but contains only finitely many non-*tock* events. [Thus it contains a record of refusals at all integer times.]

The events between two consecutive *tock*s are thought of as occurring at one of a discrete series of "moments", and the refusal set records what is refused at the point time advances, exactly in the spirit of the continuous timed failures model.

Because of this structure in which all behaviours have infinitely many *tock*s, the usual empty trace $\langle\rangle$ or $(\langle\rangle, \emptyset)$ representing a process doing nothing is replaced by $\Delta = \langle\emptyset, tock\rangle^\omega$: where time passes for ever but nothing is seen to be refused.

[4] In this paper the alphabet of events Σ will not contain *tock*, which is treated as an additional action.

While the complete Timed CSP language can be given a compositional semantics over $\mathcal{F}_{\mathbf{T}}$, to have a semantics over $\mathcal{F}_{\mathbf{DT}}$ a program needs to use only integer delays in *WAIT*s, event timings and any other places where a delay is introduced. In other words it must be an integer Timed CSP program as defined earlier.

$\mathcal{F}_{\mathbf{DT}}$ also has the **NIW** property: if an event cannot be refused before a *tock*, then it is possible after the *tock*. Intuitively, the withdrawal of the offer of some event a after a *tock* occurs because the *tock* enables some τ that changes the state. But since τ is after the *tock*, a is possible also up to that same point after *tock*. The property analogous to **CF** is the discrete forcing axiom **DF**:

$$s \in P \Rightarrow \exists s' \supseteq s. \forall a.\\ s' = s_1 \hat{} \langle X, tock \rangle \hat{} s_2 \wedge a \notin X \Rightarrow (s' \hat{} \langle a \rangle \hat{} \Delta \in P \wedge s' \hat{} \langle X, tock, a \rangle \hat{} \Delta \in P)$$

Here, $s \subseteq s'$ means that the traces (of ordinary events and *tock*s) in the two behaviours are the same, and that each refusal in s is a subset of the one at the corresponding point in s'.

DF can be paraphrased as saying that each observed behaviour of P must have arisen from an actual behaviour of the underlying machine performing the same trace, where the complement of the refusal at each *tock* were the events being offered at that point. Each such event could therefore occur either before or (because of **NIW**) after the *tock*.

We can think of s' being a complete behaviour that the the axiom forces as an extension of s, just as in the continuous case.

There are, naturally, analogues of the other three axioms of $\mathcal{F}_{\mathbf{T}}$:

NE^D $\Delta \in P$

IC^D Whenever $s \hat{} s' \in P$ and s'' has the same sequence of events as s, and with its refusals point-wise subsets of those of s, then $s'' \in P$.

NZ^D For each P there is a bound on how many events it can perform in any finite time:

$$\forall t. \exists .n. s \in traces(P) \Rightarrow \#s \upharpoonright t \leq n$$

where here $s \upharpoonright t$ means the prefix of s up to its tth *tock*.

As in the continuous case, we can create a version of **DF** without **NIW** for languages where this does not hold: the final conjunct $s' \hat{} \langle X, tock, a \rangle \hat{} \Delta \in P$ would be deleted on the RHS of the implication.

In both $\mathcal{F}_{\mathbf{T}}$ and $\mathcal{F}_{\mathbf{DT}}$, the no instantaneous withdrawal property gets in the way of the idea of determinism. If we continue to identify determinism with processes being unable, after any given trace, both to accept and refuse any event, it is clear that no process that ever withdraws an offer can be deterministic.

Definition 3.1 [ReRo99] A timed process is said to be *quasi-deterministic* when any visible event that occurs at time t *either* is the first to occur at that time and has not been refused in an interval up to t, *or* is not refused at t. Specifically:

- Over $\mathcal{F}_{\mathbf{T}}$, if $(s \hat{} \langle (a, t) \rangle, \aleph) \in P$ then *either* $t > 0$ and $end(s) < t$ and for all $\epsilon > 0$ we have $(s, \aleph \upharpoonright t \cup \{a\} \times [t - \epsilon, t)) \notin P$ *or* for all $\epsilon > 0$ we have $(s, \aleph \upharpoonright t \cup \{a\} \times [t, t + \epsilon)) \notin P$.
- Over $\mathcal{F}_{\mathbf{DT}}$, if $s \hat{} \langle a \rangle \hat{} s' \in P$ then *either* s has the form $s'' \hat{} \langle X, tock \rangle$ and $s'' \hat{} \langle X \cup \{a\}, tock, \{a\}, tock \rangle \hat{} \Delta \notin P$ or $s \hat{} \langle \{a\}, tock \rangle \hat{} \Delta \notin P$.

In each model, the first case allows a process to be quasi-deterministic even though there are traces after which an event can both be accepted and refused thanks to **NIW**. The following lemma gives characterisations of processes that are not quasi-deterministic.

Lemma 3.1 (i) *Over $\mathcal{F}_{\mathbf{T}}$ the process P is non-quasi-deterministic if and only if it has a behaviour $(s, \aleph)$ such that one of the following applies:*

(a) *There exist a and $t_2 > t_1 = end(s)$ such that both $(s \hat{} \langle (a, t_1) \rangle, \aleph \upharpoonright t_1)$ and $(s, \aleph \cup \{a\} \times [t_1, t_2))$ are in P.*
(b) *There exist a and $t_3 > t_2 > t_1 \geq end(s)$ such that both $(s \hat{} \langle (a, t_2) \rangle, \aleph \upharpoonright t_2)$ and $(s, \aleph \cup [t_1, t_3))$ are in P.*

(ii) *Over* $\mathcal{F}_{\mathbf{DT}}$ *the process* P *is non-quasi-deterministic if and only if it has a behaviour* $s^\frown\Delta$ *such that one of the following applies:*

(c) *s does not end in tock, and both* $s^\frown\langle a\rangle^\frown\Delta$ *and* $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ *belong to* P.

(d) *s has the form* $s'^\frown\langle X, tock\rangle$, *and both* $s^\frown\langle a\rangle^\frown\Delta$ *and* $s'^\frown\langle X \cup \{a\}, tock, \{a\}, tock\rangle^\frown\Delta$ *are in* P.

Proof. We show that failure of quasi-determinism implies one of the situations in the Lemma. Cases (a) and (c) relate to events a that occur either at time 0 or where another event has already occurred at the same time. These mean that **NIW** is not in force and only the second disjunct of the respective definition of quasi-deterministic can apply. In the continuous case we set $t_1 = t$ and $t_2 = t + \epsilon$, where ϵ is the example denying the "for all" in the second part of the $\mathcal{F}_{\mathbf{T}}$ part of Definition 3.1 for $(s, \aleph \upharpoonright t)$

Cases (b) and (d) relate to events that happen at time occupied by no previous events and which is > 0, meaning that we can have observed what was refused *before* the present time and *after* any previous events. In these cases the definitions of quasi-determinism tell us that one of two things happen. In the continuous case, again choose ϵ from the relevant part of Definition 3.1. Setting $t_1 = t - \epsilon$, $t_2 = t$ and $t_3 = t + \epsilon$, each of the two disjuncts implies that $(s, \aleph \cup \{a\} \times [t_1, t_3)) \notin P$, since the timed refusal in each is a subset of $\aleph \cup \{a\} \times [t_1, t_3)$. In the discrete case, the second disjunct $s^\frown\langle\{a\}, tock\rangle^\frown\Delta \notin P$ implies the first, meaning that the first (which is what is needed for (d)) holds.

The fact that (a) or (b), and respectively (c) or (d) mean a process is not quasi-deterministic in $\mathcal{F}_{\mathbf{T}}$ or $\mathcal{F}_{\mathbf{DT}}$ is also elementary. □

Some simple examples are:

- $(WAIT\,1 \,\Box\, a \to STOP);\ STOP$, which in either model can either perform a or refuse it at time 1 is quasi-deterministic.
- On the other hand

$$((WAIT\,1 \,\Box\, a \to STOP);\ STOP) \sqcap (WAIT\,1;\ ((a \to STOP) \,\Box\, (b \to STOP)) \setminus b)$$

is not because behaviours in which a on the RHS of $\sqcap$ occurs after the refusal of a at any point in $[0, 1)$ (continuous model) or before the first *tock* (discrete model) have neither of the alternative properties.

Over the continuous model $\mathcal{F}_{\mathbf{T}}$, quasi-determinism exactly captures the concept of refinement maximality.[5]

Theorem 3.1 *Over* $\mathcal{F}_{\mathbf{T}}$ *a process* P *is refinement maximal (i.e.* $Q \sqsupseteq P \Rightarrow Q = P$*) if and only if it is quasi-deterministic.*

Proof. The proof of this rests on the structural axiom **CF** quoted above. In fact (as we will show) both quasi-determinism and refinement maximality are equivalent to the following:

$$(*)\quad \begin{array}{l} \forall (s, \aleph) \in P.\, \exists! \aleph'.\aleph' \supseteq \aleph.(s, \aleph') \in P \wedge \forall\, t \geq 0.\, \forall\, a. \\ (s \upharpoonright t^\frown\langle a\rangle, \emptyset) \notin P \Leftrightarrow (\exists\, \epsilon > 0.[max\{end(s \upharpoonright t), t - \epsilon\}, t + \epsilon) \subseteq \aleph') \end{array}$$

In other words, there is a unique (i.e. $\exists!$) timed refusal $\aleph'$ associated with every trace with the property that the trace can be extended at any time by any event just when that event has either just been withdrawn in $\aleph'$ or is not refused in $\aleph'$. $\aleph'$ is thus the only possible complement of the set of events the process actually offers through the trace.

Note that the uniqueness of $\aleph'$ means that, for any fixed s, we are certain to get the same $\aleph'$ for any $\aleph$ such that $(s, \aleph) \in P$. So in particular every $\aleph$ must be a subset of the one generated by (*) for $(s, \emptyset)$.

If P does satisfy the above then it is straightforwardly quasi-deterministic. It is maximally refined because if $(s, \aleph) \in P - P'$ for some refinement P' then either $s \in traces(P')$ or not. If so we get a contradiction because the $\aleph'$ given for s in P' by **CF** necessarily omits some $\{a\} \times [t_1, t_2) \subseteq \aleph$ where no event in s appears in the given interval. **CF** then implies that $s \upharpoonright t_1{}^\frown\langle(a, (t_1 + t_2)/2)\rangle$ is a trace of P' even though it cannot be one of P. We can thus infer that the $\aleph$s associated with each trace s of P' are the same as those for s in P. So we must have $s \notin traces(P')$ and that there is therefore some shortest trace $s^\frown\langle(a, t)\rangle$ in P but not in P'. That also gives a contradiction since the trace $s^\frown\langle(a, t)\rangle$ is implied by **CF** applied to trace s.

[5] It was not equivalent to maximality in [ReRo99] because that paper used the concept of stability discussed above.

For any process $P \in \mathcal{F}_{\mathbf{T}}$ one can construct a refinement P' satisfying property (*) by induction on the length of trace: we start with $\aleph^{\langle\rangle}$ chosen by **CF** for $(\langle\rangle, \emptyset)$. The length 1 traces are then just those implied by $(\langle\rangle, \aleph^{\langle\rangle})$ under **CF**. Each such trace $s = \langle(t, a)\rangle$ gives an $\aleph^s$ implied by **CF** from $(s, \aleph^{\langle\rangle} \restriction t)$ where it can be assumed that $\aleph^s \restriction t = \aleph^{\langle\rangle} \restriction t$. We then simply continue this process inductively for longer and longer traces, and finally identify P' with the set of all $(s, \aleph)$ such that the trace s is generated at some point in this process and $\aleph \subseteq \aleph^s$.

P' satisfies property (*) by construction. It is necessarily equal to P if the latter is refinement maximal, demonstrating that maximality implies (*).

If P is quasi-deterministic, then P' omits no behaviour of P: if it did then this behaviour would differ from those picked for P' after some shortest trace s on which they agree. Whether the extra behaviour were a refusal token $\{a\} \times [t_1, t_2)$ or event (a, t) after s, it would contradict quasi-determinism using arguments similar to the above.

This concludes the proof of Theorem 3.1. □

The equivalence shown above to (*) establishes the following corollary.

Corollary 3.1 *A quasi-deterministic process in $\mathcal{F}_{\mathbf{T}}$ is completely determined by its traces. In other words, if P and Q are quasi-deterministic and have the same set of traces, then $P = Q$.*

It is natural to expect, given the above, that quasi-determinism corresponds to refinement maximality over $\mathcal{F}_{\mathbf{DT}}$ as well, but it is not true. Consider the processes

$$P_1 = a \rightarrow STOP$$

$$P_2 = ((a \rightarrow STOP) \,\Box\, WAIT\,1);\ (WAIT\,1;\ a \rightarrow STOP)$$

Over the continuous model these are not comparable in the refinement order: in fact they are both quasi-deterministic and therefore maximal. Note in particular that there are traces that the first process has but the second does not, for example $\langle(a, 1.5)\rangle$.

However over the discrete model every trace of P_1 ($\{\langle\rangle\} \cup \{\langle(a, n)\rangle \mid n \in \mathbb{N}\}$) is also one of P_2: the trace $\langle(a, 1)\rangle$ is present by **NIW**. However the first has less refusals since it does not have the behaviour $\langle\emptyset, tock, \{a\}, tock\rangle^\frown\Delta$ which the second does. Therefore P_2 is not maximal even though it satisfies the definition of quasi-determinism over $\mathcal{F}_{\mathbf{DT}}$.

In order to be refinement maximal over $\mathcal{F}_{\mathbf{DT}}$, any withdrawal of an offer must be for at least two time units.

Theorem 3.2 *Over $\mathcal{F}_{\mathbf{DT}}$, a process is refinement maximal if and only if it is quasi-deterministic and satisfies the following:*

- *If $s^\frown\langle a\rangle^\frown\Delta$ and $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ both belong to P, then $s^\frown\langle\{a\}, tock, \{a\}, tock\rangle^\frown\Delta \in P$.*

In other words, if a is withdrawn at time t then it must be withdrawn for two time units.

Proof. We establish "only if" first, then "if".

Assume that $s^\frown\langle a\rangle^\frown\Delta$ and $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ are in P but $s^\frown\langle\{a\}, tock, \{a\}, tock\rangle^\frown\Delta$ is not. We must show that P is not maximal. This follows because the absence of the last of these behaviours forces $s^\frown\langle\{a\}, tock, a\rangle^\frown\Delta \in P$ and hence $s^\frown\langle\emptyset, tock, a\rangle^\frown\Delta \in P$. Similarly, if $s'^\frown\langle X \cup \{a\}, tock\rangle^\frown v$ is any behaviour which forces the presence of $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ (with s' having the same events as s but possibly larger refusals) then necessarily $s'^\frown\langle X \cup \{a\}, tock, \{a\}, tock\rangle^\frown\Delta \notin P$ (for the presence of this behaviour would imply the presence of $s^\frown\langle\{a\}, tock, \{a\}, tock\rangle^\frown\Delta$). So $s'^\frown\langle X, tock, a\rangle^\frown\Delta \in P$.

This means that P remains a process if we delete all its behaviours which imply the presence of $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ under axiom $\mathbf{IC}^D$. In other words, we can remove the one-step refusal of a in the same way that we did in the example earlier when observing that P_1 refines P_2 over $\mathcal{F}_{\mathbf{DT}}$. P cannot have been refinement maximal, since we have deleted a non-empty set of behaviours from it.

For the "if" half of the result we have to show that a process which is quasi-deterministic with the two-step gap property set out in the statement of this result is maximal. If it were not, there would be some behaviour s that could be removed (probably along with others) and have it remain a process. We show this is impossible by induction on the (by definition of $\mathcal{F}_{\mathbf{DT}}$) finite number of non-*tock* events in s. What we actually prove by induction on this number of events is the non-existence of such an s together with (i) the fact that that the behaviour generated by axiom **DF** for a trace of each length is unique, and (ii) that the only non-*tock* events that can occur after such a trace are the ones whose presence is demanded by **DF**. So structurally the proof is closely related to the one from the continuous model.

Consider first the behaviour f_Δ which witnesses a complete refusal forced under **DF** from the null behaviour Δ in a refinement P_{-s} of P without s. We can similarly "force" the removable behaviour s in P to get f_s.

By construction f_s and f_Δ both satisfy the conditions of the RHS of **DF** but are different (for f_s implies s under $\mathbf{IC}^D$ but f_Δ does not). They will therefore respectively be of the forms $e^\frown\langle X, tock\rangle^\frown d_1$ and $e^\frown\langle Y, tock\rangle^\frown d_2$ for some (possibly empty) common prefix e and X/Y being the first places at which they differ. Without loss of generality (for the rest of this argument is symmetric in the use of f_s and f_Δ) assume that there is some $a \in X - Y$. Necessarily, then $e^\frown\langle a\rangle^\frown\Delta$ and $e^\frown\langle\{a\}, tock\rangle^\frown\Delta$ are both in P. So, by assumption, $s^\frown\langle\{a\}, tock, \{a\}, tock\rangle^\frown\Delta \in P$ also. (1)

Since $e^\frown\langle Y, tock\rangle^\frown d_2$ satisfies the RHS of **DF**, it also follows that $e^\frown\langle Y, tock, a\rangle^\frown\Delta \in P$ (though the **NIW** aspect of **DF**) and hence also $e^\frown\langle\emptyset, tock, a\rangle^\frown\Delta \in P$. (2)

The combination of (1) and (2) contradicts the quasi-determinism of P, with $s'' = e$ in that definition.

What we have in fact established here is that there is a unique extension of the null behaviour Δ under the assumptions of the Theorem, namely (i) for this base case. (ii) follows because the refusals in the discrete definition of quasi-determinism must be contained in those of the unique f_Δ, meaning that the only initial visible events to occur are those forced by f_Δ.

We have thus established that under the conditions of the Theorem, the behaviour of a process up to and including the first visible action is completely determined: it has no choice over what to offer. As in the continuous case, the step case of the recursion then just repeats the same arguments for the behaviours that extend a given $s^\frown\langle a\rangle$ up to and including the *next* visible event. This then completes the proof. □

So if we were to model the Timed CSP process

$$((a \to STOP) \,\Box\, WAIT\,1);\ (WAIT\,1;\ a \to STOP)$$

in a domain where one time unit between *tock*s is 0.5 of the one used to measure *WAIT*s we would get a refinement-maximal process, since a is withdrawn for 2 of the shortened *tock*-units.

4. The functionality of FDR

FDR is a model checker which by now has many features. That relevant to this paper is its ability to check for two properties of processes: refinement over a variety of models, and determinism. These are well known and well documented for untimed CSP. For that, the main models for calculating refinement are traces, failures and failures-divergences, the last two being equivalent for divergence-free processes.

As set out earlier, determinism means the combination of divergence freedom and the process never having both the trace $s^\frown\langle a\rangle$ and the failure $(s, \{a\})$ (i.e. it can perform s and then refuse to perform any member of the set $\{a\}$.) As well as the natural determinism check over the failures-divergences model $\mathcal{N}$, FDR can also attempt to perform a check (over the stable failures model $\mathcal{F}$) which ignores potential divergence. For divergence-free processes this always gives the same result, but there is a use relevant to this paper where divergence is sometimes possible and makes FDR fail to produce an answer. See Sect. 5 below.

All the above is well known for untimed CSP and discussed fully in [Ros10], but new capabilities allow it to do these things in the context of *integer* Timed CSP as defined earlier. This extension to CSP is reported in [ALOR12], and allows the user to mix, in a single script, integer Timed CSP and "*tock*-CSP", namely the language of untimed CSP in which the passage of time units is represented via the event *tock* that is included in programs like the members of Σ. In fact FDR's implementation of Timed CSP works by translating that language to a special form of *tock*-CSP that is semantically equivalent to it over $\mathcal{F}_{\mathbf{DT}}$.[6]

In running both Timed CSP and *tock*-CSP, FDR requires the user to apply an operator that gives internal events τ priority over the passage of time via *tock*. This operator is

$$pri(P) = priority(P, \{\}, \{tock\})$$

in the priority notation used by FDR. This is needed to achieve maximal progress as described above.

[6] The full *tock*-CSP language does not have a semantics over $\mathcal{F}_{\mathbf{DT}}$, but the dialect used for this translation does.

FDR can perform refinement checks between Timed CSP processes, where time is represented via the *tock* event, using all the usual models that it supports (traces, failures etc). These are frequently the most appropriate models for comparing a complete Timed CSP process against a specification—these are often written in *tock*-CSP—but it is important to remember that most of them are not congruences for Timed CSP: one cannot for example infer over Timed CSP that $P \sqsubseteq_T Q \Rightarrow C[P] \sqsubseteq_T C[Q]$ for a Timed CSP context $C[\cdot]$.

FDR is also capable of checking refinement in $\mathcal{F}_{\mathbf{DT}}$ between integer Timed CSP programs, which is compositional. At present this is done by using the refusal testing model embedded within FDR and a transformation on the Timed CSP processes it generates, but it may be implemented directly in future versions. It is not directly relevant to the substance of this paper, but does make compositional checking of security properties possible. Since the security properties we shall define are refinement closed, it follows that if we can establish that $C[P]$ satisfies one of them (for $C[\cdot]$ a Timed CSP context (integer in the discrete cases)) and $P \sqsubseteq Q$ in the appropriate timed failures model, then $C[Q]$ satisfies the same property.

5. Noninterference via determinism?

This ability to characterise a deterministic process even though its internal construction includes nondeterministic choice is the key to the definition of noninterference in [RWW96, Ros95, Ros97]. Almost all (the notable exception being the untimed traces model) of the behavioural models of CSP and Timed CSP enable this naturally. Given a process P with two users whose disjoint alphabets H and L partition its own, we can say that a process P can transmit no information from H to L if $\mathcal{A}_H(P)$ is deterministic, where $\mathcal{A}_H(P)$ abstracts away the behaviour of a most nondeterministic user who might control H (which should be equivalent to the nondeterministic choice of all such users). We consider all of the things the high level might do on its side of P, take the nondeterministic choice of all of them, and specify that the low level user's view must be deterministic despite that.

This is a very elegant definition, but (as with every other definition of noninterference over complex behaviours that we are aware of) it is not perfect:

- It only captures information flow that is visible in the patterns of behaviour recorded in the model being used: so if we are using the failures-divergences model a process can pass this specification despite having timing channels. (However, as remarked in [Ros06], this definition is insensitive to which of a large class of untimed models for concurrency is used.)
- It does not distinguish between nondeterminism that is causally linked to the actions of H and that which is intrinsic to P's behaviour. Even nondeterminism that is built in to help conceal H behaviour from L will mean that P is deemed insecure. Thus the definition is only exact for deterministic P; for nondeterministic P it is conservative in the sense that it never deems an insecure process secure, but might say a secure one is insecure. (As discussed in [Ros97], for example, the class of models in which $\mathcal{N}$ rests are simply incapable of making the necessary distinctions when P is nondeterministic.) Thus our conditions do not suffer from the refinement paradox—they are closed under refinement—at the penalty of treating all nondeterminism visible to L as bad.

In this paper we are addressing the first of these problems. To handle the second without admitting insecure processes as secure would require much more operational and intensional models of processes. We will discuss this further in Sect. 9.

The abstraction used should capture all the ways in which P can be influenced by the process interacting with it in H. If this interaction follows the standard CSP model then potentially that user can not only select which H action is picked when several are made available, but also whether one is selected at all. In this case the correct abstraction to use is *lazy abstraction*, defined[7] over the stable failures model $\mathcal{F}$ (in which the divergence component of $\mathcal{N}$ is replaced by one of finite traces) by

$$\mathcal{L}_H(P) = (P \parallel CHAOS_H) \setminus H$$

(The use of $\mathcal{N}$ with this formulation creates problems because it can introduce divergence that is not appropriate.)

[7] A different definition of lazy abstraction was used for defining noninterference in [RWW96, Ros95], namely $P \,|||\, RUN_H$. Determinism of this gives an equivalent definition of noninterference, but the one quoted here (taken from [Ros97]) is superior for other purposes. The interleaving definition would cause problems in the timed world because the result of applying it would breach the no-Zeno requirement.

An alternative form of abstraction called *mixed abstraction* is used when H is partitioned into two sets H_D and H_S, where the user is assumed to be able to delay the first but not the second, which are *signals* from process to user.

$$\mathcal{L}_H^{H_S}(P) = (P \parallel CHAOS_{H_D}) \setminus H$$

In this paper we will concentrate on lazy abstraction, but everything we do would work under an analogous treatment of mixed abstraction.

So the untimed definition of noninterference on which our work in this paper will be based is the following.

Definition 5.1 The process P is said to be *lazily independent* of H over the failures-divergences model $\mathcal{N}$ if $\mathcal{L}_H(P)$ is deterministic.

Numerous examples of how this definition works in characterising information flow can be found in [Ros95, RWW96, Ros97], as can results such as the demonstration that a deterministic process P is equivalent to the independent parallel composition $P_H \mid\mid\mid P_L$ where $P_L = \mathcal{L}_H(P)$ and $P_H = \mathcal{L}_L(P)$ if and only if both these processes are deterministic. In other words P is *separable* if and only if P is lazily independent of both L and H.

This immediately suggests that the way to check if a finite-state untimed process satisfies this is to ask FDR if $\mathcal{L}_H(P)$, formulated using $CHAOS_H$ as above, is deterministic. However it is not quite as simple as that, since the check can be subverted by the same divergences (resulting from infinite sequences of hidden H actions in P) that mean the definition does not work in $\mathcal{N}$. In fact these divergences can even subvert a determinism check carried out in the stable failures model $\mathcal{F}$, since FDR's algorithm to do that does not always work on a divergent process—see [Ros97]. As remarked there, one reliable method for doing this is the pair of checks

- $P \underset{H}{\parallel} STOP$ deterministic (over $\mathcal{N}$)
- $P \underset{H}{\parallel} STOP \sqsubseteq_F (P \underset{H}{\parallel} CHAOS_H) \setminus H$

The truth of this pair of checks together is equivalent to $\mathcal{L}_H(P)$ being deterministic, and do not allow an infinite sequence of H actions to cause a problem.

A second reliable method is to replace FDR's built-in check for determinism by a method that can be implemented directly in terms of the tool's refinement checking capabilities, namely to compare two copies of the process P being checked and forcing the second to follow exactly every trace that the first follows. If it never diverges and this always succeeds then P is deterministic, otherwise it is not. Since we will be adapting this idea (originally due to Lazić [Laz99]) later in this paper, we realise it below in a way easily implemented in FDR. Here *clunk* is an event that the process P does not use itself and $E = \Sigma - \{clunk\}$:

$$\begin{aligned}
CReg &= x?E \rightarrow clunk \rightarrow CReg \\
Clunking(P) &= P \underset{E}{\parallel} CReg \\
Test &= x?E \rightarrow x \rightarrow Test \\
RHS(P) &= ((Clunking(P) \underset{\{Clunk\}}{\parallel} Clunking(P)) \underset{E}{\parallel} Test) \setminus \{clunk\} \\
LHS &= STOP \sqcap x?E \rightarrow x \rightarrow LHS
\end{aligned}$$

The use of *clunk* keeps the two copies of P within one event of each other, so that each pair of events between consecutive *clunk*s come one from each copy. *Test* forces the two to follow the same trace. The specification LHS allows anything this $RHS(P)$ might do except for one process being unable to follow the other's lead causing deadlock, or P diverging.

$LHS \sqsubseteq_{FD} RHS(P)$ is then true if and only if P is deterministic, and if $\sqsubseteq_{FD}$ is replaced by $\sqsubseteq_F$ we get a test for the failures model version of determinism that is not vulnerable to the issue described above. Therefore applying the above to the $CHAOS_H$ formulation of $\mathcal{L}_H(P)$ gives our second reliable test of lazy independence.

The intuition of the deterministic low-level abstraction implying absence of information seems equally valid in Timed CSP, and indeed any of the checks for it listed above works as least as well in the discrete version implemented in FDR. Indeed the no-Zeno assumption implies that the simple formulation using the direct FDR failures or failures/divergences determinism check is guaranteed to work as hiding high-level events cannot introduce divergent behaviour.

$$(P \underset{H \cup \{tock\}}{\parallel} TCHAOS_H) \setminus H$$

can be tested for determinism where $TCHAOS_H$ is the *tock*-CSP process

$$TCHAOS_H = tock \rightarrow TCHAOS_H \,\Box\, (STOP \sqcap ?x : \Sigma \rightarrow TCHAOS_H)$$

and P has been translated into *tock*-CSP by FDR. (Bear in mind in this definition that $tock \notin \Sigma$.)

Note that *TCHAOS* violates the no-Zeno assumption, but that if P satisfies it then so does the construction for lazy abstraction above: in $P \underset{H \cup \{tock\}}{\parallel} TCHAOS_H$, the composed $TCHAOS_H$ cannot perform an infinite number of events in a finite time unless P does.

If the lazy abstraction of a Timed CSP process P is deterministic, then this does imply absence of information flow, at least information flow measurable in the model being considered. The fact that no process which ever withdraws an offer is deterministic thanks to **NIW** represents a major problem for this definition.

Given our analysis in Sect. 3, we should contemplate modifying our definition so that it is deemed free of information flow if the abstraction is either refinement maximal or quasi-deterministic. Over the continuous model $\mathcal{F}_{\mathbf{T}}$ these are the same thing, but it is as well to ask which if either is in principle the right answer.

If we believe that the model $\mathcal{M}$ we are using records all the observations that Lois might make are the ones recorded in whatever semantic model we are using, then the right answer appears to be "maximally refined". For we know that the observations she can make will be those possible for some process P_H in $\mathcal{M}$, depending on how the high-level user Hugh chooses to behave. Whatever Hugh does will be a refinement of the least refined process he can be. So if Lois's view is already maximally refined for the least refined Hugh, nothing he can do can change her view. This gives us a much stronger guarantee than simply saying that Lois's view is independent of Hugh's behaviour, because it also allows for possible variability in the system P's.

We illustrate this with an example: suppose *LEAK* is any process that passes information from Hugh to Lois, for example

$$LEAK = hugh?x \rightarrow lois!x \rightarrow LEAK$$

Now suppose that M is any process with alphabet L (implying that $\mathcal{L}_H(M) = M$ such that $M \sqsubseteq \mathcal{L}_H(LEAK)$. Then if $P = M \sqcap LEAK$ then Lois's view of the combination of P and *Hugh* will be equivalent to M no matter what process with alphabet H we pick for *Hugh*. Nevertheless the system P is allowed to behave like *LEAK* which is not secure. In fact, because M never communicates with Hugh, the latter knows that anything he communicates to P will immediately be sent to Lois.[8]

In this example $M = \mathcal{L}_H(P)$ will not be maximally refined (because $\mathcal{L}_H(LEAK)$ is not), meaning that P does not satisfy our lazy independence property. Thus insisting that $\mathcal{L}_H(P)$ is maximally refined shows that neither Hugh's decisions or the ways in which P can behave as a more refined process can affect Lois's view.

All this is, of course, very similar to the justification of the determinism-based definition of noninterference, which is not surprising. Where maximally refined processes are not deterministic this new definition requires the knowledge that whatever nondeterminism that remains cannot be resolved by whatever Hugh does and whatever internal decisions are made in P. In other words, whatever nondeterminism remains in a maximally refined process must remain in the mechanism that Lois observes. With this caveat, we can express the following re-characterisation of noninterference.

[8] If different instances of *Hugh* combined with P produce different answers, this is concrete evidence that information be passed through P, so we can certainly use such comparisons to search for covert channels. It is just that such a comparison cannot easily be justified as a complete test for information flow. We will discuss this issue further in Sect. 9.

Generalised characterisation of noninterference Suppose we have a semantic model in which refinement coincides with the reduction of the visible nondeterminism that can be resolved by implementation decisions. Then if the abstraction $\mathcal{A}_H(P)$ characterises how P appears to L in the presence of the most nondeterministic conceivable behaviour in H, we can deem P to be independent of L if $\mathcal{A}_H(P)$ is maximal in the refinement order.

It is consistent with this principle that there can be a discrete-time process with quasi-deterministic abstraction that allows information flow from H to L, and we can construct one as follows. Recall P_1 and P_2 as defined before Theorem 3.2. With *tock*-time unit 1, with a an event in L and h an event in H and b a further event, where both h and b take the same time (say d) to complete, we can then define:

$$R = (h \rightarrow P_2 \,\Box\, b \rightarrow P_1) \setminus \{b\}$$

This process is certain to perform either h or the hidden b in the first time step, and if Lois ever sees a refused after the delay d but before a has occurred, then she will know h has occurred. The natural lazy abstraction of R is just $WAIT\, d;\ P_2$, which is quasi-deterministic.

So we have concrete evidence that quasi-determinism of the abstraction over discrete models does not always mean absence of information flow, and a powerful argument that under certain assumptions the refinement maximality of the abstraction does guarantee it. Nevertheless we will find in Sect. 6 that quasi-determinism over discrete models can be useful nevertheless.

5.1. Abstraction over timed failures

In order to give substance to the specifications of noninterference implied above, we must formulate abstraction over the continuous and discrete timed failures models. We concentrate on lazy abstraction but remark that mixed abstraction poses no problems other than getting the lazy part of it right: high level actions that cannot be delayed by Hugh are still hidden.

We start with $\mathcal{F}_{\mathbf{DT}}$. We want $\mathcal{L}_H(P)$ to represent how P looks to an observer unable to see alphabet H on the assumption that there is a user interacting with P in H with the full capability of offering subsets of events to the process that vary (a) when an event occurs and (b) with time.

Simply translating the untimed formulation to Timed CSP:

$$(P \underset{H}{\parallel} CHAOS_H^-) \setminus H \quad \text{where}$$
$$CHAOS_H^- = STOP \sqcap ?x : H \rightarrow CHAOS_H^-$$

brings a number of problems.

- If some events in H take more than 0 time to complete, $CHAOS_H^-$ defined like this is not as general as it should be since it cannot immediately follow up such an event with another, even though we can imagine Hugh as a parallel process that can. *The natural way to solve this problem is to define $CHAOS_H^-$ in an environment where all events take zero time.*
- If some events in H take 0 time (which they will if we follow the solution above) then the recursion for $CHAOS_H^-$ is not time guarded and the process can perform an infinite number of events in a finite time. *This means that $CHAOS_H^-$ is not a proper Timed CSP process. However it is still reasonable to regard the parallel composition $P \underset{H}{\parallel} CHAOS_H^-$ as one since $CHAOS_H^-$ can perform no more actions than P does. So this is more of an apparent problem than a real one, just as in the tock-CSP formulation of lazy abstraction above in terms of $TCHAOS_H$.*
- More subtly, imagine the situation where the $CHAOS_H^-$ process defined above has resolved its nondeterministic choice in the first time step, but no H action occurs before the first *tock*. The operational semantics of CSP give it no way of changing its mind for the second time step. At the level of timed failures, the semantics of this $CHAOS_H^-$ does not contain the behaviour $\langle H, tock, h\rangle^\frown\Delta$ for $h \in H$. It is because this definition is deficient in this way that we have given it the superscript $-$.

An efficient definition that does work (still subject to the assumption that events in it take zero time, and that it must be put in parallel with a no-Zeno process) is

$$CHAOS^A_H = (?x : H \to CHAOS^A_H) \rhd (WAIT\,1;\ CHAOS^A_H)$$

where $\rhd$ is the asymmetric choice operator that initially offers the choice of the events of its left-hand argument with a τ that takes it to its right-hand argument. ($P \rhd Q$ is equivalent to $(P \Box a \to Q) \setminus \{a\}$ for an event a not appearing in either P or Q.) Thus $CHAOS^A_H$ can perform any sequence of H events in a given time unit but may at any time opt not to perform any more before the next *tock*. This version never offers events from H in a stable state, which could be problematic in some contexts, but will not be when events from H are hidden as they are in abstraction: that explains the superscript A (for abstraction).

So our definition of lazy abstraction over $\mathcal{F}_{\mathbf{DT}}$ will be

$$\mathcal{L}_H(P) = (P \underset{H}{\parallel} CHAOS^A_H) \setminus H$$

which gives us our first concrete timed definition of noninterference.

It is also possible to use the *tock*-CSP process $TCHAOS_H$ defined earlier, which is also able to change its decisions about whether or not to offer H events each *tock*. In FDR is is equivalent to do the latter to an integer Timed CSP process when it has been translated to *tock*-CSP, or to apply the discrete Timed CSP definition before that translation.

Definition 5.2 A process defined in integer Timed CSP is *1-independent of* H if $\mathcal{L}_H(P)$ (defined as above) is maximally refined in $\mathcal{F}_{\mathbf{DT}}$.

The reason for the 1 in this name will become apparent in Sect. 6. In examining examples of timed noninterference we will largely restrict ourselves to examples which satisfy *untimed* noninterference, such as

$$\begin{array}{lcl} P & = & l \to l \to P \Box h \to LS \quad \text{where} \\ LS & = & l \to LS \end{array}$$

This, seemingly, just offers the event $l \in L$ for ever, possibly interrupted by a single $h \in H$ after an even number of ls. Since l is always on offer this satisfies the untimed definition of noninterference in the usual direction, but note that L can pass information to H by choosing an odd or even number of ls. So over untimed CSP, $\mathcal{L}_H(P)$ is deterministic and $\mathcal{L}_L(P)$ is nondeterministic.

For the definition of this P to be time guarded, we need l to take non-zero time to complete. However $\mathcal{L}_H(P)$ is only maximally refined over $\mathcal{F}_{\mathbf{DT}}$ if h takes zero time, for otherwise there is a period after h when l is refused in a way in which it would not have been it h had not happened. So for example the abstraction will have the behaviours $\langle l \rangle^\frown \Delta$ and $\langle \{l\}, tock \rangle^\frown \Delta$ if h take more than 0 time to complete. That would not be compatible with being maximally refined. On the other hand, if h does take time 0 (so that P is willing to communicate l immediately after h), the abstraction is equivalent to LS.

This example teaches us an expected lesson: *if H and L share access to a sequentially defined process P, considerable care is necessary to eliminate all timing channels from H to L.*

It is also interesting to note that if h takes one time unit then $\mathcal{L}_H(P)$ is quasi-deterministic even though non-maximal, but that if either h takes at least two units, or we use the model $\mathcal{F}_{\mathbf{DT}}$ with the *tock* unit 0.5, then it is not quasi-deterministic.

We will see further examples of timed noninterference analysis later.

The problem with defining abstraction over the continuous model $\mathcal{F}_{\mathbf{T}}$ is that time moves forward continuously rather than discretely. The process $CHAOS^A_H$ depends crucially on there being a *next* time at which things happen. The best way of defining lazy abstraction over $\mathcal{F}_{\mathbf{T}}$ is as a primitive operator over this model:

$$\mathcal{L}_H(P) = \{(s \setminus H, \aleph \cup \aleph') \mid (s, \aleph) \in P \wedge \aleph' \subseteq H \times [0, \infty)\}$$

In other words P is allowed to perform any behaviour at all, but any offers in H it makes are not visible to the outside world. The assumption here is that at times when $\aleph$ does not contain the whole of H, the abstracted copy of H is allowed to refuse any such events that P offers. It is interesting to contrast this with the definition of hiding which insists that P is always forced to perform as many H events as it can—namely when time progresses the whole of H is hidden:

$$P \setminus H = \{(s \setminus H, \aleph) \mid (s, \aleph \cup H \times [0, \infty)) \in P\}$$

It should not be too hard to see that our discrete time definition of $\mathcal{L}_H(P)$ using $CHAOS_H^A$ can be re-written in a form similar to the continuous one above. There is a strong reason to code the discrete definition directly in the Timed CSP notation that does not apply in the continuous case, namely that the discrete model has been implemented in FDR.

Definition 5.3 A process defined in (general) Timed CSP is lazily independent of H over $\mathcal{F}_{\mathbf{T}}$ if $\mathcal{L}_H(P)$ (defined as above) is maximally refined, or equivalently quasi-deterministic, in $\mathcal{F}_{\mathbf{T}}$.

This continuous definition gives the same results for the simple process P that we studied in the discrete case above, namely that is free of H to L information flow if and only if h takes zero time.

An obvious question we can ask at this stage is whether the two definitions always coincide like this for integer Timed CSP. Unfortunately the answer to this is "no", with the problem arising because of the distinction between maximally refined and quasi-deterministic processes over $\mathcal{F}_{\mathbf{DT}}$.

We saw above that halving the time interval to 0.5 can turn our first example of a quasi-deterministic process that is not maximal into a maximal one. We might ask if a similar transformation might work in general, or provide a bridge between the continuous and discrete definitions: might it be the case that an integer Timed CSP is continuously noninterfering if and only if it is with a discrete interpretation with a 0.5 *tock*, or perhaps for a $1/n$ *tock* for other n that is either fixed or process dependent? The answer to this is *no*, as we show below, though a rather similar question will have positive answer in Theorem 6.3.

To build the counter-example to the question above, we remark first that that any process actually constructed as the parallel composition of two processes P_L and P_H with alphabets respectively L and H, not communicating at all, is unable to pass information from H to L or vice versa: $P = P_L \;|||\; P_H$. Both our definitions of lazy abstraction give $\mathcal{L}_H(P) = P_L$ (as is also the case in untimed CSP: see [Ros97]), and so the question of whether our definitions of noninterference are satisfied by such a P comes down to whether P_L is maximally refined when interpreted in the discrete and continuous models respectively. If our two definitions of noninterference coincided, or if they did for some $1/n$ *tock*, then they would have to agree on this question also. We therefore construct a process which is maximal in the continuous time model but in no discrete one, no matter how large n is chosen in picking *tock* to be of length $1/n$.

Assume both $l1$ and $l2$ are low level events that take time 0 to complete.

$$
\begin{aligned}
P_L &= (Q_1 \;|||\; Q_2) \underset{\{l2\}}{\parallel} (l2 \rightarrow STOP) \quad \text{where} \\
Q_1 &= (l1 \rightarrow ((l2 \rightarrow STOP) \,\Box\, WAIT\,1);\; STOP) \\
Q_2 &= WAIT\,4;\; l2 \rightarrow STOP
\end{aligned}
$$

P_L offers $l2$ for one time unit after $l1$ occurs, and the offer is then withdrawn. However a second and indistinguishable offer of $l2$ is always made at time 4 unless the other one has been taken up first. Note that the parallel composition with $l2 \rightarrow STOP$ ensures that only one $l2$ can occur in total, meaning that if both $l2$s are available at the same time, the nondeterminism over which occurs has no visible consequences.

Over the continuous model this process is quasi-deterministic and hence maximal: the offer of $l2$ is withdrawn if $l1$ occurs early enough and later reappears thanks to Q_2. Over the discrete model it is still quasi-deterministic but not maximal in the case where the gap between the end of the first offer and time 4 is one *tock*. Decreasing the interval between the *tock*s to 0.5 (or any other reciprocal) does not help here as it did in an earlier example, because the event $l1$ can always happen at the time that will leave the gap at one *tock*.

So for this example no time interval for *tock*s that makes the *WAIT*s in the program an integer multiple of it will make this P_L maximal.

If we regard the continuous semantics as definitive and the discrete ones as approximations, it is comforting to note that the discrete model of noninterference appears to differ only in the direction of being more conservative. We will be able to justify this formally in the next section thanks to digitisation.

6. Digitisation: playing with time

The theory of digitisation was introduced by Henzinger, Manna and Pnueli in [HMP92] as a way of proving properties about continuous systems (specifically, timed automata) by analysing discrete approximations. It was adapted for Timed CSP by Ouaknine [Oua01, Oua02] who showed that one can prove certain properties of systems over the continuous model $\mathcal{F}_{\mathbf{T}}$ by demonstrating analogous properties of discrete approximations. In particular he showed that every integer Timed CSP program has the property of being *closed under digitisation* and therefore refines any specification that is *closed under inverse digitisation* (certain, but not all integer Timed CSP programs) if and only if the refinement holds over $\mathcal{F}_{\mathbf{DT}}$.

In this section we will examine how properties such as quasi-determinism and noninterference behave under digitisation. Our objective will be to find a way of verifying that an integer Timed CSP program is lazily independent of H by analysis over $\mathcal{F}_{\mathbf{DT}}$.

In order to make the following analysis easier we will assume that the only sources of time delays in our programs are integer *WAIT* statements: one can recode delays that occur when events happen or recursion unfolds into this form if required.

We characterise digitisation as the application of certain sorts of transformations that change the times of the actions (visible, invisible and evolutions through time) that occur in process's execution in standardising way but which preserve the validity of the execution. They are formalised in terms of the operational semantics of Timed CSP defined by Schneider in [Sch00]. Our characterisation is slightly more general than that in [Oua01, Oua02], because we need the possibility of retiming to non-integers for our later applications, but its intuition and proof details are essentially the same as Ouaknine's.

Before introducing digitisation we consider more general *retimings* of operational semantics: monotonic (but not necessarily strictly so) mappings from $\mathbb{R}^+$ to itself intended to preserve the validity of operational semantics when applied to the beginning and end of all times of operational semantic transitions. Such transitions are visible and invisible actions, which are instantaneous, and timed evolutions such as $P \overset{t}{\rightsquigarrow} Q$ whose end is t after its beginning.

One cannot arbitrarily re-time such behaviours because of maximal progress: an event that happens at the time a τ action becomes available can only be re-timed to the moment the corresponding τ becomes available in the transformed behaviour. A retiming is *valid* if this it true. Given our assumption that *WAIT*s are the only source of delays, the only way in which a τ (or any other action) can become available through a time evolution is when a *WAIT* expires somewhere within the program.

We restrict our attention to *integer periodic* retimings ϕ which map each integer time to itself, and which map each $n + x$ (for $0 < x < 1$) to $n + \phi(x)$ (with $\phi(x)$ necessarily being in the closed interval [0, 1]). The fact that we are considering only *integer* Timed CSP means that if we retime the start of a *WAIT* n from $k + x$ to $k + \phi(x)$ then the end of it is retimed from $k + x + n$ to $k + \phi(x) + n$, which is of course the time that our *WAIT* n ends when its start is retimed. In general a Timed CSP term that does not immediately have a τ can evolve through any time up to and including the first moment a *WAIT* statement within it elapses. So one can prove (following [Oua01]) via a combination of structural induction on a term with mathematical induction on the number of actions and time evolutions that have occurred that:

- After k steps the original and retimed programs are in the same state except for the exact times remaining on non-zero *WAIT*s. If the remaining time is zero on a *WAIT* in the original, then it also is in the retimed one. A non-zero time remaining on a *WAIT* in the original program can become zero in the retimed one when the original program's time and the time when that *WAIT* expires map to the same value. In this case the original behaviour certainly has an action between these two times, so the retimed one has an action *at* the same (retimed) moment.
- The original and retimed programs have exactly the same set of actions available except where the retimed version has a *WAIT* retimed to 0 as discussed above, in which case the retimed one has an additional τ. Time evolutions are available up to and including the minimum remaining time on any *WAIT* each of the original and retimed states respectively contain.
- In any case (i) any action that the original performs now is valid in the retimed state (ii) if the retimed process is obliged by maximal progress to perform an action now then the retimed behaviour contains an action at the same time.
- If the present time is t and the original and retimed programs have a *WAIT* that elapses respectively in times x and y, then $\phi(t + x) = t + y$.

We can conclude:

Theorem 6.1 1. *An integer periodic retiming ϕ is valid on an integer Timed CSP program P.*

2. *If $(s, \aleph)$ is in the $\mathcal{F}_{\mathbf{T}}$ representation of P, then so is $(\phi(s), \phi(\aleph))$, where ϕ acts on the times of events and the end points of the half-open intervals during which $\aleph$ is constant. (Note that if $\aleph$ includes some $X \times [x, y)$ where $\phi(x) = \phi(y)$ then $\phi(\aleph)$ retains no "memory" of this since $[\phi(x), \phi(y))$ is then empty.)*

The proof of the second part of this result follows from the fact that the value of a process in $\mathcal{F}_{\mathbf{T}}$ can be obtained by formally observing the execution paths through the operational semantics. The transformation on the execution path created by an allowable retiming ϕ has exactly the stated effect on the timed failures observable of that path.

We define a *digitisation* to be an integer periodic retiming whose image in any interval $[n, n + 1]$ is finite. In other words, a digitisation transforms all of the actions in a behaviour to ones that happen at members of a pre-determined discrete set of times.

Ouaknine, in developing the above ideas, concentrates on digitisations that map every time t to the integer above or below it, and specifically $[t]_\epsilon$ for $0 < \epsilon \leq 1$ that maps $n + x$ to n or $n + 1$ depending on whether $x < \epsilon$ or $x \geq \epsilon$. For our purposes we need a little more flexibility. Identify $[t]_\epsilon$ with $[t]_{\langle\epsilon\rangle}$ and allow the subscript, in general, to be any finite, nonempty and strictly monotonic sequence of numbers in the range $(0, 1]$.

Definition 6.1 $[t]_{\langle\epsilon(1),\ldots,\epsilon(n)\rangle}$ is the retiming that maps $r + x$ ($r \in \mathbb{N}$, $0 \leq x < 1$) to r if $x < \epsilon(1)$, to $r + 1$ if $x \geq \epsilon(n)$ and to $r + \frac{m}{n}$ if $\epsilon(m) \leq x < \epsilon(m + 1)$ for $1 \leq m < n$.

In the following, $frac(x)$ is defined to be the unique number $0 < y \leq 1$ such that $x - y$ is an integer. So in particular $frac(n) = 1$ for $n \in \mathbb{N}$. $\lfloor x \rfloor$ is the greatest integer less than or equal to x: its integer part. The following technical lemma, which follows directly from the definition above, is needed for the proof of Lemma 6.2 below.

Lemma 6.1 *Suppose $frac(x) = \epsilon(i)$ for $1 \leq i \leq n$. Then*

(i) *If $y < x$ then $[y]_{\langle\epsilon(1),\ldots,\epsilon(n)\rangle} < [x]_{\langle\epsilon(1),\ldots,\epsilon(n)\rangle}$.*

(ii) $\lfloor x \rfloor < [x]_{\langle\epsilon(1),\ldots,\epsilon(n)\rangle}$

Lemma 6.2 *If $0 \leq t_0 < t_1 \cdots < t_n$ are $n + 1$ values in $\mathbb{R}^+$ in order, then we can choose $0 < \epsilon 1 < \cdots < \epsilon k \leq 1$ for some $k \leq n$ such that $[\cdot]_{\langle\epsilon 1,\ldots,\epsilon k\rangle}$ maps $t_0, \ldots, t_n$ to distinct values, necessarily separated by at least $\frac{1}{k}$.*

Proof. Arrange the values $frac(t_0), \ldots, frac(t_n)$ into sorted order and remove any duplicates, obtaining a list $\langle\epsilon 0, \epsilon 1, \ldots, \epsilon k\rangle$ for some $k \leq n + 1$.

We consider the following cases: when $k \leq n$ and when $k = n + 1$. In the first, at least two of the t_i have the same fractional part. Note that any such pair (being at least 1 apart) map to different values under any integer periodic retiming. In this case we can simply let the digitisation be $[\cdot]_{\langle\epsilon 1,\ldots,\epsilon k\rangle}$. What we have to show is that this maps all the t_i to distinct values. By monotonicity, if this fails there must be a consecutive pair t_i and t_{i+1} that map to the same value. This is impossible because, unless $frac(t_i) = frac(t_{i+1})$ (the case we have already shown map to distinct values), the fractional parts of $[t_i]$ and $[t_{i+1}]$ are themselves distinct and included amongst the ϵi, so Lemma 6.1 (i) applies.

In the second case we know that the fractional parts of all the t_i are different. In this case let $\underline{\epsilon}$ be the sequence $\langle\epsilon 0, \epsilon 1, \ldots, \epsilon k\rangle$ with $frac(t_0)$ (which is not necessarily $\epsilon 0$) deleted. This list thus contains $frac(t_i)$ for every $i > 0$. Suppose that $[\cdot]_{\underline{\epsilon}}$ maps two (without loss of generality) consecutive t_i to the same value. These cannot be t_0 and t_1 because *either*

- t_0 and t_1 have the same integer part and $frac(t_0) < frac(t_1)$ and $frac(t_1)$ is a member of $\underline{\epsilon}$ so Lemma 6.1 (i) applies, *or*
- $[t_0]_{\underline{\epsilon}} \leq \lfloor t_1 \rfloor < \lfloor t_1 \rfloor_{\underline{\epsilon}}$ The strict inequality here is because $frac(t_1)$ is a member of $\underline{\epsilon}$, and thanks to Lemma 6.1 (ii).

This completes our proof. □

We will require this result for $n = 2$, where there are three times $t_0 < t_1 < t_2$, and the resulting retiming separates them by intervals of at least $\frac{1}{2}$. In the case where $n = 1$ the result captures the essential property of digitisation with a single parameter and which maps all numbers to integers; we will use this a number of times.

Ouaknine establishes a crucial connection between the discrete and continuous semantics of integer Timed CSP. It is easy to see a relationship between $\mathcal{F}_{\mathbf{DT}}$ behaviours and integer $\mathcal{F}_{\mathbf{T}}$ behaviours—ones where everything (i.e. events and changes in $\aleph$) happens at an integer time—given one of the former s we map it to $\psi(s) = (u, \aleph)$, calculated as follows:

- s takes the form $s_0{}^\frown\langle X_1, tock\rangle{}^\frown s_1{}^\frown\langle X_2, tock\rangle{}^\frown\cdots$ where each s_i is a finite trace in Σ^*, $X_i \subseteq \Sigma$ and all but finitely many s_i are empty. Let u_i have the same events as s_i but with each event paired with i: for example if $s_3 = \langle a, a, b\rangle$ then $u_3 = \langle (a, 3), (a, 3), (b, 3)\rangle$.
- Define u to be the concatenation of all the u_i (i.e. $u_0{}^\frown\cdots{}^\frown u_r$ where u_r is the last non-empty u_i).
- Define $\aleph = \bigcup_{i=1}^{\infty} X_i \times [i-1, i)$.

Thus $\psi(s)$ is a timed failure that is a natural model for s.

Theorem 6.2 [Oua01] *For any integer Timed CSP process P, the integer behaviours in its $\mathcal{F}_{\mathbf{T}}$ semantics are exactly $\psi(s)$ as s ranges over its $\mathcal{F}_{\mathbf{DT}}$ semantics.*

It is clear that the continuous time semantics of any Timed CSP program P are isomorphic[9] to those of kP for $k \geq 1$ an integer, which is the same program except that all $WAIT\,n$ are transformed to $WAIT\,(kn)$ provided we scale all behaviours of kP by dividing all the times by k. Since kP is an integer Timed CSP program if P is, we can deduce the following lemma. Here a *half integer* $\mathcal{F}_{\mathbf{T}}$ behaviour is one where all events and changes in $\aleph$ occur either at integers or $n + \frac{1}{2}$ for an integer n.

Lemma 6.3 *For any integer Timed CSP process P, the half-integer behaviours in its $\mathcal{F}_{\mathbf{T}}$ semantics are exactly the scalings by $\frac{1}{2}$ of $\psi(s)$ as s ranges over the $\mathcal{F}_{\mathbf{DT}}$ semantics of $2P$.*

This is exactly the result we need to create a decision procedure for noninterference defined over $\mathcal{F}_{\mathbf{T}}$.

Theorem 6.3 I. *An integer Timed CSP process P that is lazily independent over $\mathcal{F}_{\mathbf{T}}$ (i.e. has a quasi-deterministic lazy abstraction) has a quasi-deterministic lazy abstraction over $\mathcal{F}_{\mathbf{DT}}$.*

II. *An integer Timed CSP process P is lazily independent of H (judged over $\mathcal{F}_{\mathbf{T}}$) if and only if $\mathcal{L}_H(2P)$ is quasi-deterministic when judged over $\mathcal{F}_{\mathbf{DT}}$ (or equivalently if $\mathcal{L}_H(P)$ is quasi-deterministic when judged over $\mathcal{F}_{\mathbf{DT}}$ in which the length of one tock is 0.5).*

Proof. We first prove I.

If $\mathcal{L}_H(P)$ is not quasi-deterministic over $\mathcal{F}_{\mathbf{DT}}$ then one of conditions (c) and (d) from Lemma 3.1 (ii) applies. From this, and the definition of $\mathcal{L}_H$, we get two cases:

- If (c) (the case where the behaviour visible to L prior to the nondeterminism does not end in *tock*) then for some L event l, P has behaviours over $\mathcal{F}_{\mathbf{DT}}$ of the forms $s_1{}^\frown\langle l\rangle{}^\frown\Delta$ and $s_2{}^\frown\langle\{l\}, tock\rangle{}^\frown\Delta$ where deleting the H events in s_1 and s_2 leaves the behaviour s. Without loss of generality (because of **IC**) we assume that all the pre-*tock* refusals in these are $\emptyset$. Theorem 6.2 then tells us that P has the $\mathcal{F}_{\mathbf{T}}$ behaviours $(\psi(s_1)_1, \{l\} \times [n, n+1))$ and $(\psi(s_2)_1{}^\frown\langle(l, n)\rangle, \emptyset)$ where there are n tocks in each of s, s_1 and s_2 and $\psi(s)_1$ takes the non-*tock* events in s and makes a timed trace by attaching the number of *tock*s preceding each as its time.
  Here ψ is the map defined earlier from $\mathcal{F}_{\mathbf{DT}}$ behaviours to integer $\mathcal{F}_{\mathbf{T}}$ ones, so $\psi(s)_1$ extracts just the timed trace from this.
  The definition of $\mathcal{L}_H$ over $\mathcal{F}_{\mathbf{T}}$ then tells us that $\mathcal{L}_H(P)$ has the behaviours $(\psi(s)_1, \{l\} \times [n, n+1))$ and $(\psi(s)_1{}^\frown\langle(l, n)\rangle, \emptyset)$, meaning it is not quasi-deterministic since it is easy to see that $end(\psi(s)_1) = n$ by construction and therefore case (a) of Lemma 3.1 (i) applies to the continuous semantics of $\mathcal{L}_H(P)$.
- The second case is where (d) applies. A very similar argument then shows that (b) applies to the continuous semantics of $\mathcal{L}_H(P)$.

We can deduce that $\mathcal{L}_H(P)$. This completes the proof of I.

[9] Here, the term "isomorphic" can be understood either in terms of transition systems or in terms of $\mathcal{F}_{\mathbf{T}}$. In the first case, it is clear that there is a bijection β from the states of the operational semantics of P to those of kP, in which, for states U and V, $U \xrightarrow{x} V$ for $x \in \{\tau\} \cup \Sigma$ if and only if $\beta(U) \xrightarrow{x} \beta(V)$, and where $U \overset{t}{\rightsquigarrow} V$ if and only if $\beta(U) \overset{k\times t}{\rightsquigarrow} \beta(V)$ for $t \geq 0$. In the case of $\mathcal{F}_{\mathbf{T}}$, the timed failures of kP are obtained from those of P by multiplying all the times in the timed traces and refusals by k. This plainly provides a definition of kP as an operator over $\mathcal{F}_{\mathbf{T}}$, and we will use it like that occasionally.

This implies that if $\mathcal{L}_H(2P)$ is quasi-deterministic in $\mathcal{F}_{\mathbf{T}}$ then it is in $\mathcal{F}_{\mathbf{DT}}$, which establishes one of the two implications required for II. The fact that $\mathcal{L}_H(2P)$ being quasi-deterministic is equivalent to $\mathcal{L}_H(P)$ having that property (by Lemma 6.3 and the easy result that (over $\mathcal{F}_{\mathbf{T}}$) $\mathcal{L}_H(2P) = 2\mathcal{L}_H(P)$).

So we need prove that if $\mathcal{L}_H(P)$ is quasi-deterministic over $\mathcal{F}_{\mathbf{DT}}$ then it is over $\mathcal{F}_{\mathbf{T}}$. Assuming that $\mathcal{L}_H(P)$ is not quasi-deterministic over $\mathcal{F}_{\mathbf{T}}$ gives us the two options of Lemma 3.1 (i), of which the second is more interesting.

- In case (b), we can assume that δ has been chosen sufficiently small so that, in the two behaviours $(s_1 \hat{} \langle (l, t) \rangle, \emptyset)$ and $(s_2, \{l\} \times [t - \delta, t + \delta))$ that P must have for some $l \in L$ with $s_1 \setminus H = s_2 \setminus H$ and $end(s_1 \setminus H) < t$, no event of s_1 or s_2 occurs in $[t - \delta, t + \delta)$ except at t. We can also assume that $\delta \leq \frac{1}{2}$.
  We can now invoke Lemma 6.2 with $t_0 = t - \delta$, $t_1 = t$ and $t_2 = t + \delta$ to get a valid digitisation of integer Timed CSP that maps these three times to three consecutive (thanks to $\delta \leq \frac{1}{2}$) members of the series $\langle \frac{n}{2} \mid n \in \mathbb{N} \rangle$. Applying this to $(s_1 \hat{} \langle (l, t) \rangle, \emptyset)$ and $(s_2, \{l\} \times [t - \delta, t + \delta))$ and then scaling by 2 tells us that $2P$ has behaviours $(s_1' \hat{} \langle (l, t') \rangle, \emptyset)$ and $(s_2', \{l\} \times [t' - 1, t' + 1))$ where all events occur at integer times, t' is an integer, and $s_1' \setminus H = s_2' \setminus H$. Theorem 6.2 and the definition of lazy abstraction over $\mathcal{F}_{\mathbf{DT}}$ then tells us that $\mathcal{L}_H(2P)$ is not quasi-deterministic over $\mathcal{F}_{\mathbf{DT}}$. [Note that both s_1' and s_2' can have some events happening at the image of t under the digitisation.]
- The case where (a) applies is simpler than the above because we can consider an interval $[t, t + \delta)$ rather than $[t - \delta, t + \delta)$, and so only two points are involved in the digitisation, meaning that we do not need the factor of 2.

This completes the proof of Theorem 6.3. □

This result is very powerful in the context of noninterference since it tells us that a particular discrete model, which observes an integer Timed CSP process only at discrete times, is sufficient to prove that there is no information flow to an observer who can observe it at any and all times.

It suggests the following discrete definition of noninterference:

Definition 6.2 For $k \in \mathbb{N} - \{0, 1\}$, we define the integer Timed CSP P to be k-lazily independent of H if kP is quasi-deterministic when judged in $\mathcal{F}_{\mathbf{DT}}$.

The multiplier 2 in the formulation and proof of Theorem 6.3 could have been replaced by any integer $k \geq 3$, with the choice of t_i $(i > 2)$ being arbitrary at the point where Lemma 6.2 is used. This means that P's lazy independence of H, judged over $\mathcal{F}_{\mathbf{T}}$ is equivalent to P being k-lazily independent of H for every $k > 1$. Thus all these conditions are equivalent to each other:

Theorem 6.4 *For integer Timed CSP, the conditions k-lazy independence are all equivalent for* $k \geq 2$.

The second part of Theorem 6.3 is not true without the multiplier 2. Consider $P_3 = P_1 \sqcap P_2$, where P_1 and P_2 are defined before Theorem 3.2. Setting $L = \{a\}$ and $H = \emptyset$ (so that $\mathcal{L}_H(P_3) = P_3$ in both models), we know that the the discrete abstraction is equivalent to P_2 and quasi-deterministic. The continuous one, on the other hand, can perform a at time 1.5 and refuse it over the interval $[1, 2)$.

The example given in Sect. 5.1 of a process P_L which is quasi-deterministic over all models, but not maximal in any discrete one (which is equivalent to kP_L not being maximal in $\mathcal{F}_{\mathbf{DT}}$ for any $k > 0$) tells us something rather unexpected about the discrete characterisations of noninterference. This is that processes (such as $P_H \mid\mid\mid P_L$ for any P_H that only communicates in H) can be k-independent ($k > 1$) without $\mathcal{L}_H(kP)$ being maximal in $\mathcal{F}_{\mathbf{DT}}$. This seems at odds with the analysis given earlier that non-maximality leads to information flow. This is not in fact an issue because $\mathcal{L}_H(kP)$ is maximal amongst the images of kP' as P' varies over integer Timed CSP processes.

The reason for the doubling of the "metronome" in the discrete approximation used to decide quasi-determinism derives from needing a discrete witness to the three distinct times that exemplify one sort of failure of quasi-determinism. It it worth noting that if we restrict ourselves to processes that never withdraw offers (so quasi-determinism and determinism are the same) then it is not necessary to use the doubling, because we can decide whether or not the continuous semantics of an integer Timed CSP process are deterministic using the natural rather than doubled discrete model. This is because the failure of a continuous process being deterministic shows up (after the timed trace s) in *two* times $end(s) \leq t_1 < t_2$ where both $s \hat{} \langle (a, t_1) \rangle \in traces(P)$ and $(s, \{a\} \times [t_1, t_2))$ is a timed refusal. This means that it is not necessary to use the extended form of digitisation we used above, and in fact we get the following.

Theorem 6.5 *For an integer Timed CSP process P:*

(a) *P is deterministic in the continuous semantics if and only if it is deterministic in the ordinary discrete semantics.*

(b) $\mathcal{L}_H(P)$ *is deterministic over* $\mathcal{F}_{\mathbf{T}}$ *if and only its discrete semantics is deterministic over* $\mathcal{F}_{\mathbf{DT}}$.

Here, (a) is a trivial consequence of (b)—one can set $H = \emptyset$. The proof of (b) follows the structure of that of Theorem 6.3 II except that, since the failure of determinism is witnessed by a timed trace, a visible event and *two* (rather than three) distinct points in time, there is no need to use the multiplier 2 as Lemma 6.2 can be used for $n = 1$.

We stated, at the end of Sect. 5, that 1-lazy independence appeared to be more conservative than the continuous time definition. We are now in a position to justify this.

Theorem 6.6 *If P is an integer Timed CSP process that is 1-lazily independent, then it is lazily independent when judged over* $\mathcal{F}_{\mathbf{T}}$.

Proof. We know that such a P has $\mathcal{L}_H(P)$ both quasi-deterministic over $\mathcal{F}_{\mathbf{DT}}$ and with the property that any withdrawal of an offer extends to at least two time units, as defined in the statement of Theorem 3.2. Suppose it was not quasi-deterministic over $\mathcal{F}_{\mathbf{T}}$. Then again we have to consider the two options of Lemma 3.1 (i). The first (where the offending event a happens at time 0 or immediately after another) implies non-quasi-determinism of $\mathcal{L}_H(P)$ over $\mathcal{F}_{\mathbf{DT}}$ relatively straightforwardly: if $(s^\frown\langle(a, t)\rangle, \aleph \restriction t)$ and $(s, \aleph \cup \{a\} \times [t, t + \delta))$ are the offending pair of behaviours with δ chosen small enough that in the second behaviour P has no action in the interval $(t, t + \delta)$, the digitisation $[\cdot]_{frac(t+\delta)}$ maps them to a pair of integer behaviours which (thanks to Theorem 6.2) prove the existence of a pair of discrete behaviours of $\mathcal{L}_H(P)$ satisfying the conditions in Lemma 3.1 (c).

It is when the offending continuous behaviours satisfy the conditions of Lemma 3.1 (b) (so the offending event happens after an interval when nothing has occurred) that we need the extended withdrawal quality of $\mathcal{L}_H(P)$ over $\mathcal{F}_{\mathbf{DT}}$. Without loss of generality (as argued below thanks to the healthiness conditions of $\mathcal{F}_{\mathbf{T}}$ and the congruence between operational and denotational semantics) we can assume that the $\mathcal{F}_{\mathbf{T}}$ behaviours of $\mathcal{L}_H(P)$ that deny its quasi-determinism are $(s^\frown\langle(a, t)\rangle, \emptyset)$ and $(s, \{a\} \times [t_1, t_2))$ where

(i) $t_1 < t < t_2$

(ii) $t_1 \geq end(s)$

(iii) $t_2 \leq t + 1$

(iv) Subject to constraints (i), (ii) and (iii), $[t_1, t_2)$ is the maximal interval derived from some operational behaviour of $\mathcal{L}_H(P)$ which, after s, refuses a and performs no visible actions during $[t_1, t_2)$.

To construct t_1 and t_2, take any behaviour of $\mathcal{L}_H(P)$ with trace s and which refuses a in an interval $[t - \delta, t + \delta)$ for $t - \delta \geq end(s)$. P will then have $(s_P, \{a\} \times [t - \delta, t + \delta))$ for some s_P with $s_P \setminus H = s$, and a behaviour β of its operational semantics to witness this, where β extends forward in time to $t + 1$. There may, be nonempty intervals during $[end(s), t + 1) - [t - \delta, t + \delta)$ where β offers a. Delete these and let $[t_1, t_2)$ be the remaining component that contains t.

We need to consider two sub-cases:

(a) Where $t_1 = end(s)$, and so a is refused through the interval $[end(s), t_2)$.

(b) Where $t_1 > end(s)$, and so P (and hence also $\mathcal{L}_H(P)$) offered a for an interval ending at t_1.

In case (a), consider the digitisation $[\cdot]_{frac(t_2)}$. This certainly maps t and t_2 to consecutive integers, and maps t_1 and t to integers that may or may not be distinct. In the case that $[t_1]_{frac(t_2)} < [t]_{frac(t_2)}$ we get case (d) of Lemma 3.1 for the corresponding discrete behaviour. If $[t_1]_{frac(t_2)} = [t]_{frac(t_2)}$ we find that the digitisations of $(s^\frown\langle(a, t)\rangle)$ and $(s, \{a\} \times [t_1, t_2))$ yield case (c) of Lemma 3.1 as the a occurs at $end(s')$ (s' being the digitised version of s).

In case (b), we contemplate two further sub-cases: where $t - t_1 \geq 1$ and where $t - t_1 < 1$. In the first of these we can use the same digitisation as in (i), because now $[t_1]_{frac(t_2)} < [t]_{frac(t_2)} < [t_2]_{frac(t_2)}$ and we get case (d) of Lemma 3.1. If $t - t_1 < 1$, consider the digitisation $[\cdot]_{frac(t)}$. This is guaranteed to map t_1 and t to consecutive integers, but may map t and t_2 to the same value. We know that each of $(s^\frown\langle(a, t_1)\rangle, \emptyset)$, $(s, ^\frown\langle(a, t)\rangle)$ and $(s, \{a\} \times [t_1, t_2))$ is a behaviour of $\mathcal{L}_H(P)$, from which we can conclude that each of $s'^\frown\langle a\rangle^\frown\Delta$, $s'^\frown\langle\emptyset, tock, a\rangle^\frown\Delta$ and $s^\frown\langle\{a\}, tock\rangle^\frown\Delta$ is a discrete behaviour of $\mathcal{L}_H(P)$.

At last we can use the two-*tock* withdrawal property of this abstraction that forms part of the characterisation of maximality in Theorem 3.2. We use this to conclude from the first and third of these behaviours that $s^\frown\langle\{a\}, tock, \{a\}, tock\rangle^\frown\Delta$ is also a discrete behaviour of $\mathcal{L}_H(P)$. Together with the second of the behaviours at the end of the previous paragraph, this proves that $\mathcal{L}_H(P)$ is not quasi-deterministic over $\mathcal{F}_{\mathbf{DT}}$.

Thus, by a rather complex case analysis, we can conclude that if P is not lazily independent over $\mathcal{F}_{\mathbf{T}}$, then it is not 1-lazily independent. □

7. Deciding noninterference using FDR

It is possible to test for both quasi-determinism and refinement maximality over $\mathcal{F}_{\mathbf{DT}}$, using FDR.

7.1. Deciding quasi-determinism

We deal first with quasi-determinism, developing a variant of Lazić's test for determinism. As in that, we compare two copies of a process, this time checking that for every visible action a the first copy of a process P performs, a second one *either* cannot refuse it after the same trace *or* a occurred immediately after a *tock* and the second copy was unable to refuse a prior to the corresponding *tock*.

We therefore have to keep two copies of P running—say P_1 and P_2 where P_1 performs actions and P_2 has to prove for each one that it cannot refuse them appropriately. From the description above it is clear that P_2 has to follow the same trace—including *tock*s—as P_1 but that it has sometimes to be at an earlier time than P_1. Namely, when P_1 performs an event after *tock*, we have to test whether P_2 can refuse it before, and if so after exactly the same *tock*. Therefore P_1 and P_2 are not synchronised on *tock*: in fact the latter can be 0, 1, and sometimes 2 tocks behind P_1.

We can immediately deduce that the check we devise to check for this is not going to be constructed in Timed CSP, but rather in *tock*-CSP with multiple *tock* events. As with the check for determinism that we are adapting, our check will take the form:

$$LHS \sqsubseteq_F RHS(P)$$

where $RHS(P)$ this time consists of two copies renamed so that they give different names to *tock*, and where they strictly alternate their non-tock events. The two copies are put in a testing harness, and because of the nature of the latter we give the "follower" copy of P two separate names for *tock*. Overall

$$\begin{array}{lcl}
RHS(P) & = & \Pi((first(P) \underset{\{turn1, turn2\}}{\parallel} second(P)) \setminus \{turn1, turn2\}) \underset{Events}{\parallel} QDTest \\
first(P) & = & P[\![^{tock1}/_{tock}]\!] \underset{E}{\parallel} FReg \\
second(P) & = & P[\![^{tock2,\, tock2'}/_{tock,\, tock}]\!] \underset{E}{\parallel} SReg \\
FReg & = & ?x : E \to turn2 \to turn1 \to FReg \\
SReg & = & turn2 \to ?x : E \to turn1 \to SReg
\end{array}$$

where *QDTest* is a testing process and Π a priority operator that we will describe below. *Events* is the set of all visible events, both Σ and the various flavours of *tock* used here. The synchronisation with and between *FReg* and *SReg* ensures that the two copies strictly alternate non-*tock* events, with *first*(P) performing the first.

The testing process has the following states:

$$\begin{array}{lcl}
QDTest & = & ?x : E \to x \to QDTest \\
 & & \sqcap\ tock1 \to QDTest' \\
 & & \sqcap\ STOP \\
QDTest' & = & tock1 \to tock2 \to QDTest' \\
 & & \sqcap ?x : E \to QDTest''(x) \\
 & & \sqcap\ STOP \\
QDTest''(x) & = & (x \to STOP \,\Box\, tock2' \to x \to QDTest) \\
 & & \sqcap\ tock2 \to x \to QDTest
\end{array}$$

These are explained:

- The tester is in state $QDTest$ when $first(P)$ and $second(P)$ have performed equal numbers of $tock$s, and their non-$tock$ traces are equal. Necessarily this is when the most recent communication of $first(P)$ was not $tock1$, because the tester deliberately then holds $second(P)$ back.
- $QDTest'$ is when the two have performed equivalent traces except that $first(P)$ has moved ahead by one $tock1$, which was the most recent event it has performed. If $first(P)$ again performs $tock1$ then $second(P)$ must perform (as it will certainly be able to) $tock2$ so it is still one behind. If $first(P)$ performs any other event x then we must check that $second(P)$ obeys the refusal requirements on it, and moves to $QDTest''(x)$ to check this.
  Neither of theses first two states insists that $first(P)$ is able to do anything—hence the inclusion of $STOP$ in the nondeterministic choice.
- $QDTest''(x)$ is when $second(P)$ has fallen behind $first(P)$ by $tock$ followed by x. It now has to do two things: check that $second(P)$ offers (i.e. cannot refuse) x either before or after its next $tock$, and also ensure that the tester is in the right state to check P's behaviour on longer traces. The latter happens automatically in other states, but requires more care here.
  For the first of these tasks, it must create an error state when $second(P)$ can refuse x both before and after P's *tock on the same execution*. This is not possible using a non-prioritised check over stable failures model $\mathcal{F}$. It could have been done with the refusal testing model $\mathcal{RT}$, but in this presentation we use priority.
  The left-hand branch of the $\sqcap$ in $QDTest''(x)$ is responsible for this part of the check. It initially offers either x or $tock2'$ to $second(P)$, overall we apply the priority operator Π to the system, which is defined to give $tock2'$ lower priority than every other action, all others being equivalent. That means that $tock2'$ can only happen when x is refused by $second(P)$.
  If x does occur in that state then $second(x)$ has passed the test created when $first(P)$ performed x. However the two copies of P have performed different traces ($tock$ and x in opposite orders) and so are permitted to behave differently. We therefore do not carry on with the test from this point on: hence this trace leads to $STOP$ in $QDTest''(x)$ (and the LHS process we define below).
  If $tock2'$ has occurred then $second(x)$ fails the test unless it accepts x in its post-$tock2'$ state. However if it does perform the x we can carry on the check because the two P's have now performed the same trace.
  If we were simply to have done the above, then some future behaviours of P would not get tested. These are the ones where x is guaranteed to be available in $second(P)$ before $tock2'$, for the priority relation then means that $second(P)$ never performs it after. This problem is solved by the second branch of $QDTest''(x)$'s $\sqcap$. That offers just $tock2$ rather than the choice of $tock2'$ and x, which brings $second(P)$ into a state where

  – If is not *obliged* to be able to perform x.
  – However we know that in at least one execution it can perform x after $tock2$, since after all the other copy of P has already performed x on the same trace.

  So we can carry out the continuing test on this branch, confident in the knowledge that P's behaviour after all possible traces will now be explored.

The reason why we have used $tock2$ and $tock2'$ is so the specification can tell which of the two testing branches has been followed. If $tock2'$ has occurred it will insist that x is offered while after $tock2$ it will not.

It is interesting to note that we have carried out two separate tests on $second(P)$ by using the $\sqcap$ operator between processes that perform them.

The specification against which this is checked in $\mathcal{F}$ is then

$$
\begin{aligned}
LHS &= STOP \sqcap (?x : E \to x \to LHS) \sqcap (tock1 \to LHS') \\
LHS' &= STOP \sqcap (tock1 \to tock2 \to LHS') \sqcap (?x : E \to LHS''(x)) \\
LHS''(x) &= (x \to STOP) \sqcap (tock2' \to x \to LHS) \sqcap (tock2 \to (STOP \sqcap x \to LHS))
\end{aligned}
$$

Here, the states correspond in an obvious way to the states of $QDTest$. This specification is guaranteed to be trace refined by RHS since $QDTest$ refines it, so the only way it can fail is when the required offers are not made. These correspond to the various failures of quasi-determinism discussed above.

So we may test an integer Timed CSP process P for independence over $\mathcal{F}_{\mathbf{T}}$ by running the check

$$LHS \sqsubseteq_F RHS(pri((CHAOS^A_H \underset{H}{\parallel} 2P) \setminus H)$$

where $2P$ and $CHAOS^A_H$ are as described above. Here *pri* is the time-priority function that represents the boundary between Timed CSP (inside it) and *tock*-CSP (outside it).

7.2. Maximality over $\mathcal{F}_{\mathbf{DT}}$

The check set out in the previous section is all that is required to test for all our noninterference conditions other than 1-lazy independence. For that, in addition to quasi-determinism, we need to check the two-step withdrawal property. Specifically, we want to be able to flag as non-maximal processes which have behaviours of the forms $s^\frown\langle a\rangle^\frown\Delta$ *and* $s^\frown\langle\{a\}, tock, a\rangle^\frown\Delta$.

This requires a Lazić-style check comparing two copies of P with the same overall shape as the one in the last section, but slightly simpler as we do not have to allow one of the copies of P to move a *tock* ahead of the other. Overall

$$\begin{array}{lcl} RHS'(P) & = & \Pi'((((first'(P) \underset{\{turn1, turn2, tock\}}{\parallel} second'(P)) \setminus \{turn1, turn2\}) \underset{E\cup\{tock\}}{\parallel} MaxTest) \setminus E) \\ first'(P) & = & (P \underset{Events}{\parallel} TCHAOS_E) \underset{E}{\parallel} FReg \\ second'(P) & = & P \underset{E}{\parallel} SReg \end{array}$$

with *FReg* and *SReg* as before and E the set of non-*tock* events used by P.

Here, the first copy of P is put in parallel with $TCHAOS_E$ to ensure that, under the outermost hiding of E, it can still do the same timed traces as P: maximal progress does not force any of this copy's actions since $TCHAOS_E$ can delay them.

The role of *MaxTest* is to copy what the first instance of P does to the second, and if the second refuses one of these communications for a time unit the offer is still made after the next *tock*. We get an error if the action then occurs.

$$\begin{array}{lcl} MaxTest & = & tock \rightarrow MaxTest \\ & = & ?a : E \rightarrow MaxTest'(a) \\ MaxTest'(a) & = & a \rightarrow MaxTest'' \\ & & \Box\ tock \rightarrow a \rightarrow error \rightarrow STOP \\ MaxTest'' & = & tock \rightarrow MaxTest \\ & = & \Box ?x : E \rightarrow x \rightarrow MaxTest'' \end{array}$$

Here, *error* is a new event outside E. The priority operator Π' is just the usual time priority operator *pri* that gives priority to internal actions over *tock*. All actions of $RHS'(P)$ other than *tock* and *error* become τ thanks to the hiding.

error can happen just when, after the two copies of P have performed the same trace, the first copy performs an action and the second refuses it for the rest of the same time unit, but then accepts the action after one further *tock*. The fact that the second copy must refuse the event to get this far is guaranteed by $pri = \Pi'$.

To check for *error* all we need to do is to see if $RHS'(P)$ trace-refines $TOCKS = tock \rightarrow TOCKS$.

A quasi-deterministic process P over $\mathcal{F}_{\mathbf{DT}}$ will satisfy $TOCKS \sqsubseteq_T RHS'(P)$ is and only if it is maximal in the same model.

7.3. Pragmatics

Running two copies of an implementation process in parallel to check for noninterference properties is potentially expensive since it means that in the worst case the state space of $RHS(P)$ is quadratic in that of P. We can reduce this problem by using either or both of the following techniques.

- We can use an FDR compression operator on P before applying RHS to it. Because the definition of RHS involves priority, this must be a compression that is valid inside the FDR *prioritise* operator: at the time of writing, by far the best option is (divergence-respecting) weak bisimulation *wbisim* as described in [Ros10]. *wbisim* is a recent addition to FDR.
- Following the first of the two alternatives presented earlier for reliably checking $\mathcal{L}_H(P)$'s determinism over untimed models, we can break the check for the abstraction's quasi-determinism over $\mathcal{F}_{\mathbf{DT}}$ into the same two parts:

  (a) $P \underset{H}{\parallel} STOP$ is quasi-deterministic

  (b) $P \underset{H}{\parallel} STOP \sqsubseteq \mathcal{L}_H(P)$ where refinement is judged over $\mathcal{F}_{\mathbf{DT}}$.

  This is attractive because in the absence of any H actions, the process being checked here for quasi-determinism is potentially significantly smaller than $\mathcal{L}_H(P)$, of which only one copy is used in (b).

In the case studies below, we will find that making (a) above true is sometimes challenging. In that case (b) alone makes (as discussed earlier) a useful but incomplete weak check for noninterference.

8. Case studies

In this section we present two related case studies, one of a sequential process and one of a parallel system. Both are intended to multiplex high and low level communications through a common medium. They indicate potential sources of timing channels in shared systems as well as possible cures. Throughout this section we assume that events take unit time to complete: $et(a) = 1$ for all $a \in \Sigma$.

8.1. Sequential shared medium

In both our examples we will imagine (as in the corresponding untimed examples in Section 12.4 of [Ros97]), that *Hugh* is sending messages to *Henry* and *Lois* is sending them to *Leah*

The following is a simple sequential process that communicates such messages:

$$\begin{aligned} TM(C) \quad = \quad & send?x : (S - \{x \mid (x, m') \in C\})?m \to TM(C \cup \{(x, m)\}) \\ & \Box \, (\Box_{(x,m)\in C} \, rec!dual(x)!m \to TM(C - \{(x, m)\}) \end{aligned}$$

where *Leah* and *Lois*, and *Hugh* and *Henry* are two pairs of duals and the initial value of C is just $\emptyset$. C contains the data presently in the medium: pairs of the sender x and the message being sent to $dual(x)$. As untimed CSP the above definition would be equivalent to the interleaving of two separate one-place buffers, one at each security level, and would satisfy every conceivable independence condition between H and L.

This is not true for Timed CSP unless the delays attached to H events are 0 (meaning that the process would be capable of behaviour violating the no-Zeno requirement). In reality the sharing of a resource like this is always likely to create a *timing channel*: the fact that *Hugh* is sending *Henry* a message will delay *Lois* from sending one to *Leah* if that takes any time at all. This is a simple example of the most common sort of timing channel: L sees the timing effects of H's consumption of system resources.

Our timed noninterference check easily identifies this problem. Note that checking it with all event times set to 2 is equivalent to checking $2TM(\emptyset)$ because there are no other sources of delay in this program.

One way of preventing information flow though systems is partitioning whatever resource gives rise to a potential channel between the levels. We can do this in the present example by only permitting high and low events at disjoint sets of times in such a way that whenever a high-level event occurs its delaying effect is over by the time that L might again perform an event.

With a process like TM we can imagine either redesigning it to a sequential process with the above quality, or modelling an operating system component that schedules access to it. We can realise the latter by building a scheduler that is placed in parallel with our system, synchronising on all events, which in different phases makes and withdraws offers in L and H. Assume there are constants LO, LB, HO, HB representing the length of the offer of low events, the break after this before high events are allowed, and the same two for high. Then we can

create our scheduler:

$$\begin{aligned} LOW \quad = \quad & (?x : L \to STOP) \triangle WAIT\,LO);\ WAIT\,LB; \\ & ((\mu\, p.?x : H \to p) \triangle WAIT\,HO);\ WAIT\,HB;\ LOW \end{aligned}$$

Putting this in parallel with TM creates a process lazily independent of H provided that $HB \geq 1$.

Note that we have allowed an arbitrary number of H-actions, and only one L-action, per time slot. That is because allowing multiple L actions creates nondeterminism itself by interaction with the scheduler: when some action is just becoming available when the time-out fires, it may or may not be offered. One can handle this in one of two ways:

- If we were to replace $?x : L \to STOP$ by $\mu\, p.?x : L \to p$ above the system would satisfy only the weaker noninterference specification

  $$P \underset{H}{\parallel} STOP \sqsubseteq \mathcal{L}_H(P)$$

  but this is not an absolute guarantee of independence.
- We can allow multiple phases of L actions for each of H.

Both of these are illustrated in the file that implements this case study.

8.2. Parallel implementation

There are various proposals in Section 12.4 of [Ros97] for how to solve the same shared channel problem in an untimed context using a network where, in addition to the channel itself, there are processes which act as intermediaries between it and each of the four users. Some of these failed and some succeeded. One which succeeded was using flow control to ensure that the central medium never gets blocked: *Leah*'s and *Hugh*'s terminal processes do not accept a second input until they receive an acknowledgement that the first has been delivered.

$$\begin{aligned} TLois \quad &= \quad send.lois?x \to in.lois!x \to out.lois.Ack \to TLois \\ THugh \quad &= \quad send.hugh?x \to in.hugh!x \to out.hugh.Ack \to THugh \\ TLeah \quad &= \quad out.leah?x : T \to rec.leah!x \to in.leah.Ack \to TLeah \\ THenry \quad &= \quad out.henry?x : T \to rec.henry!x \to in.henry.Ack \to RHenry \\ Medium \quad &= \quad in?s?x \to out!dual(s)!x \to Medium \end{aligned}$$

The terminal processes use the same external channels as the sequential model above, namely *send* and *rec*, but use the new channels *in* and *out* to communicate with the internal process *Medium*. In the complete model, *in* and *out* are hidden, so as in the first example, H will consist of the *send* and *rec* events with high-level users, and similarly for L.

This combination satisfies any reasonable untimed noninterference condition, but interpreted as Timed CSP it does not satisfy our timed ones for much the same reasons as above.

There are at least two ways one might set about putting this right:

(i) The whole ought to satisfy noninterference if we replace the *Medium* process above with a process that satisfies timed independence.

(ii) We could set out to find a solution which addresses the timing of the system as a whole.

The first of these represents sound design, and can reasonably be argued whenever (as in our case) the processes interacting with a noninterfering core are non-interacting (i.e. separated) parallel processes.

This can be realised by replacing *Medium* with the process derived in the previous section (subject to changing its channel names to *in* and *out*), and provided the system is actually constructed in this way this would lead to a secure system. It does not (at least composed in the way we did) lead to a system which satisfies the timed lazy independence property, because (like one plausible solution we discussed in the previous section) it fails to be quasi-deterministic even when H does nothing at all. It did, however, satisfy the weak noninterference condition $P \underset{H}{\parallel} STOP \sqsubseteq \mathcal{L}_H(P)$ as in the earlier case.

The solution we found that looks at overall system design was based on the fact that we need the time it takes to transport a message from *Lois* to *Leah* to be deterministic and independent of H activity, and similarly the time from delivery at *Leah* until *Lois* next being able to send another. One way of achieving this is to allow the low processes in effect, the right to book a time at which their message will be delivered and ensure that the medium is not used by anything else at that time.

An easy way of enabling this is to modify the medium so that it accepts every message twice and only delivers on the second occasion.

$$BM = in?x?m \rightarrow in!x!m \rightarrow out!x!m \rightarrow BM$$

The assumption here is that we will ensure that the second input by BM from L will be at a time that depends deterministically on whatever action by *Lois* or *Leah* instigated it, even though the first input may not be.

We can achieve this by the following re-programming of the terminal processes above into Timed CSP, where D and D' are suitably chosen delays.[10]

$$\begin{aligned}
TLoisT &= send.lois?x \rightarrow \\
&\quad ((in.lois!x \rightarrow SKIP) \,|||\, (WAIT(D);\ in.lois!x \rightarrow SKIP));\\
&\quad out.lois.Ack \rightarrow TLoisT\\
TLeahT &= out.leah?x : T \rightarrow rec.leah!x \rightarrow \\
&\quad ((in.leah.Ack \rightarrow SKIP) \,|||\, (WAIT\,D;\ in.leah.Ack \rightarrow SKIP));\ TLeahT\\
THughT &= send.hugh?x \rightarrow in.hugh!x \rightarrow in.hugh!x \rightarrow \\
&\quad out.hugh.Ack \rightarrow WAIT\,D';\ THughT\\
THenryT &= out.henry?x : T \rightarrow rec.henry!x \rightarrow in.henry.Ack \rightarrow \\
&\quad in.henry.Ack \rightarrow WAIT\,D';\ THenryT
\end{aligned}$$

The point about this is that the delays $WAIT\,D'$ in the high-level processes must create the guarantee that within $D-1$ units of starting to try to communicate with the medium, they succeed. One example that works within our timing assumptions is $D = 4$, $D' = 1$.

The result then satisfies the full timed lazy independence specification.

The approach here differs from the one in the previous section in that there is no pre-arranged partition of the resource that the two levels share (i.e. the central medium process), but rather we ensure that the low level process can always get hold of enough of it relative to the reduced and delayed transmission that our model permits to it. The exact share of the central resource that L obtains is then concealed from L itself.

An interesting consequence of this style is that no offer made to L is ever withdrawn in this construction. So in fact whenever the timing is chosen to make the abstracted system quasi-deterministic it is also deterministic. In this case, as observed earlier, we can infer the continuous result from the discrete one without doubling the metronome.

9. Other approaches to timed noninterference

There is not nearly as much literature on timed theories of noninterference as there is on untimed theories. We will here discuss two approaches to timed noninterference that appear reasonably close to ours.

Foccardi, Gorrieri and Martinelli have extended their own approach to untimed noninterference to timed process algebras in [FGM00, FGM03], much as we have here and in [HuRo06]. They consider only discrete time semantics. Analysis of the relationship of their conditions to ones of our type was previously presented in [HuRo06, Hua10], and detailed analysis of the connections between the corresponding untimed conditions can be found in [For99, FRR99]. Below we summarise the comparison with the present paper.

The timed process algebra they present (tSPA) is based on CCS, but in terms of time modelling is closer to *tock*-CSP than to discrete Timed CSP. It is like the former in that the time event (which they term *tick*) is included

[10] It is important to realise that the terminal processes are, in this example, integral components of the system—which certainly is not noninterfering without them. These delays are chosen by the system implementer to ensure both noninterference and the right balance of performance for the two levels. For example with all events set to take 1 time unit to complete, $D = 4$ and $D' = 1$ eliminate all information flow from H to L, but not in reverse, as illustrated in the accompanying CSP file `partimex.csp`.

explicitly like *tock*. They modify the choice operator in the same way as is done when modelling Timed CSP in *tock*-CSP (previously proposed, for example, in [Oua01, Sch00]), namely having time not resolve a choice and impose the property (related to this modification) of *time determinacy*[11] which is not in general true in *tock*-CSP through it is in discrete Timed CSP. We remark that the condition of being *weakly timed alive* that they define, and which is required for their noninterference definitions, is true of discrete Timed CSP processes in general. (It is not true in general of tSPA or *tock*-CSP processes.) It follows that the noninterference conditions of [FGM03] can be tested unconditionally on discrete Timed CSP via the latter's operational semantics.

The fact that *tick* is included explicitly in tSPA means that, like *tock*-CSP, it does not have the property **NIW**. This because, unlike in Timed CSP, processes react directly to "clock" signals: the process $a.TICKS + TICKS$ (where $TICKS = tick.TICKS$ is analogous to the *tock*-CSP process *TOCKS*) will withdraw the offer of a on the first *tick*.

Therefore if we the extract discrete timed failures from a weakly timed alive tSPA process by observing the timed LTS, the resulting set will be in the expanded version of $\mathcal{F}_{\mathbf{DT}}$ with the weakened **DF** (without **NIW**) that we envisaged earlier. We believe that the appropriate noninterference property in the philosophy espoused by this paper would be that the lazy abstraction is *deterministic*, since in the expanded model we should re-establish the connection between maximality and determinism. Being weakly timed alive would be just as important as in [FGM03], since lazy abstraction would not make sense without it. We leave these hypotheses for analysis in future work. Noninterference in terms of the determinism of an abstraction has been studied in the similar setting of *tock*-CSP in [Hua10].

In common with their work on untimed calculi, the authors of [FGM00, FGM03] choose LTS-based specifications of noninterference based on the idea of comparing the views of a low-level user when different high-level users are present. They have two levels of specification: the stronger *NDC* states that, given a process P and a set of events H, the encapsulated composition of P with every possible process with events H are equivalent. The difficulty of verifying this leads to the weaker *SNNI* which is the property that hiding H events in P leads to a process that is equivalent to restricting (i.e. preventing) H events.

The chosen notion of equivalence in [FGM03] is timed bisimulation (meaning weak bisimulation over a timed LTS), leading to conditions *tBNDC* (the main specification) and a pair of approximations in terms of *SNNI*, namely the more liberal property *tBSNNI* and *tSBSNNI*, which is defined to mean that every reachable state of P satisfies *tBSNNI*. *tSBSNNI* is a more restrictive property than *tBNDC*.

As is the case in the untimed world [For99] it has been shown [Hua10] that our definitions of noninterference *imply tSBSNNI*. For untimed conditions, Huang conjectures an analogue of the untimed theorem (from [For99]) that for deterministic processes, the abstraction-based properties he considers are *equivalent* to *tSBSNNI*. (Both Huang and Forster bridge this gap via the intermediate property of Strong Local Noninterference, which is a strengthening of $(t)SNSNNI$.)

There is a philosophical difference between the style of conditions proposed in the present papers and those of [FGM00, FGM03]. This has already been well documented in [HuRo06, For99, FRR99]. Our conditions, being closed under refinement, neither admit as secure processes that have insecure refinements nor ones where H has the opportunity to resolve nondeterminism in L's view provided that L gets the same range of nondeterminism in any case. *tBNDC* can, in some circumstances, allow both sorts of process. So, for example, the Timed CSP process

$$\begin{aligned} P \quad = \quad & h?x \to l!x \to WAIT\,1;\ P \\ & \rhd_0 \left(\sqcap_{x \in T}\, l!x \to WAIT\,1;\ P\right) \end{aligned}$$

(with all events taking 0 time to complete and T being the finite type associated with the channels h and l) satisfies *tBNDC* even though every item communicated by H is transmitted directly to L. We can imagine H sending a message statistically by sending whichever member of T he wishes repeatedly, and L recording the one she hears most often.

This process can be refined by making the second line

$$\rhd_0\ l!c \to WAIT\,1;\ P$$

for c a fixed member of T, where the insecurity is indisputable since H can pass a clear-text message to L provided he avoids using c. Of course tSPA has no theory of refinement, but this example shows that *tBNDC* is inconsistent

[11] Time determinacy means that a process can have at most one successor under the *tick* action in its LTS.

with the Timed CSP view of refinement as it suffers from the refinement paradox, and in any case allows some processes whose security is debatable.

This must be balanced against the reasonable criticism of our conditions that they exclude processes that display nondeterminism (or here, more accurately, behaviour that is not refinement maximal) to L, even when there is no causal link to what H might have done. We believe that in order to develop a theory of noninterference with neither deficiency one would require models (abstract, operational or both) in which the causes and nature of nondeterminism were much more explicit than in traditional models of concurrency. In particular, one would need ways to distinguish between potentially refinable nondeterminism, nondeterminism that is introduced by the actions of H, and nondeterminism that is intrinsic to the operation of P and cannot be refined.

The second piece of work that we compare ours with is that of Barbuti, de Francesco, Santone and Tesei [BFST02, BaTe03] on noninterference in the world of timed automata [AlDi94]. They choose a basic model of noninterference that is closely related to NDC, except that they parametrise it with the maximum rate of H actions permitted. Timed automata have explicit clocks rather than $WAIT\,n$ statements or *tick*s or *tock*s. They do not assume maximal progress, but rather (in this presentation) an eventual progress condition. We find it interesting that the form of timed automata in these papers excludes the idea of a state invariant (a condition on clocks that must apply in a given state). The latter is closely related to maximal progress, since it can force actions to occur, and also to the concept of a *time-stop*, namely a state in which time cannot progress and which is therefore self-contradictory. Time-stops are a phenomenon of *tock*-CSP and other timed calculi, but not of the original language of Timed CSP or the one used in this paper. They have been introduced into some versions of Timed CSP, for example the one of [OuWo03], where it was demonstrated that finite-state Timed CSP, extended in this way and also encompassing one unboundedly nondeterministic construct and the construct DIV, meaning infinitely many τ actions at one time, has equivalent expressive power to *finite-state closed timed* ϵ*-automata*.

Here, *closed* means that all clock constraints are closed, and so use non-strict inequalities $\leq$ rather than allowing both $\leq$ and $<$. So in particular, if a constraint is met by a series of times t_n that converge to a given time t^*, then the constraint holds at t^* too. The automata of [OuWo03] have non-urgent invisible transitions ϵ, as in [BFST02, BaTe03], but do have invariants. The theory of digitisation which we have used in the present paper, was originally created for closed timed automata. The authors of [OuWo03] assert

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

It is therefore reasonable to believe that the Timed CSP language of the present paper (without time-stops) must be close in expressive power to the invariant-free language of automata in [BaTe03], provided the latter is restricted to the case of closed clock constraints. Timed automata with closed clock constraints naturally satisfy the full form of the **CF** axiom (with **NIW**), since an action cannot be available at all times before time t without also being available at time t, at least under the no-Zeno assumption made in [BaTe03]. We therefore believe both that the noninterference specifications proposed in this paper will apply essentially unchanged to the natural abstractions of closed timed automata into the continuous model $\mathcal{F}_{\mathbf{T}}$, and that our development of a digitisation theory in Sect. 6 is likely to be extensible to this alternative arena.

The relationship between the abstraction-based maximality condition and the one in [BaTe03] is likely to resemble that discussed earlier in this section between timed lazy independence and tBNDC. Since the one of [BaTe03] asserts that the behaviour of a system with H-action banned is equivalent to the behaviour with them allowed (at no more than a given rate), there is also a striking similarity with the second part (b) of our factorisation of lazy (timed) noninterference into two parts at the end of Sect. 7.3. But that must be the subject of future work.

The idea of creating noninterference conditions in which the high-level user is restricted to a particular rate of events is an interesting one and would be easy to incorporate within the framework of this paper by using a modified form of lazy abstraction: $CHAOS^A(H)$ would be replaced by a process that had such a restriction built in.

10. Conclusions

We have shown in this paper how definitions of noninterference previously developed for untimed CSP can be adapted to Timed CSP. In doing so we have given new insights into the structures of both the discrete and continuous timed failures models and in particular their refinement-maximal members.

We have developed the idea that, where refinement corresponds to reduction of nondeterminism, specifying that the low-level abstraction is *maximally refined* (i.e. as deterministic as possible) is the right specification of noninterference in some circumstances, including the timed models.

Ouaknine's theory of *digitisation* for Timed CSP has been generalised, and we were able to show that in order to establish continuous-time noninterference for integer Timed CSP, it is sufficient to do so for a discrete model in which the time step *tock* is 0.5 of the units used to represent delay in the program under consideration.

More generally, we have shown how to map any behaviour of an integer Timed CSP process with $n+1$ events happening at distinct times, to another in which these times remain distinct but are now of the form $\frac{k}{n}$ for $k \in \mathbb{N}$. This suggests that one might use digitisation to establish the equivalence of the existence of such behaviours over $\mathcal{F}_{\mathbf{T}}$, for integer Timed CSP process P, and the existence of them over $\mathcal{F}_{\mathbf{DT}}$ for nP. This remains a topic for future work, as does the possibilities of our refined notion of digitisation for Timed Automata.

We were able to create FDR checks which decide our noninterference conditions for finite-state processes, and applied them to some simple case studies. From these case studies we can conclude that timed noninterference can be decided, at least on small examples, quickly and efficiently.

It was impressed on us, in creating case studies, that creating Timed CSP systems that act deterministically or quasi-deterministically is not always easy. This should not be surprising when a number of self-timed (as opposed to clock-driven) systems interact, but is of course an issue when our noninterference condition expects us to eliminate most or all nondeterminism from systems with no high-level behaviour (formalised as $P \underset{H}{\parallel} STOP$).

We have showed how a weaker noninterference specification (in fact identical in structure to the fault tolerance specification proposed in [Ros97], and similar to the noninterference condition proposed for timed automata in [BaTe03]) can apply in such circumstances. We have found it able to capture timing channels that exist in systems, but unfortunately there are situations where information flow will not be captured. Further practical research is needed on how frequently the strong noninterference specification has to be weakened in this way. This might necessitate further theoretical work in understanding for which sorts of system it, or some variant, might be sufficient. A good alternative, investigated in [Hua10, HuRo06], might be timed variants of Forster's Local Noninterference (LNI) conditions [For99, FRR99]. Further research would be needed to understand these over continuous Timed CSP and to implement timed versions of these in FDR. Some theory akin to those of [Mor06, McIMo10] may also be possible, though the question of what one must be ignorant of seems rather less tangible in the world of process algebras as opposed to models based on assignable state.

Section 9 showed that in all likelihood our conditions would make sense in the world of timed automata, with further work required to formalise the connections.

We believe that noninterference analysis will become increasingly important thanks to the advent of Cloud computing, in which software and data belonging to multiple parties use common hardware. When two applications, one of which may be specifically designed for gathering information, are sharing an implementation platform, it will be necessary for security to show that information cannot leak from one to the other.

In addition to the type of conditions presented in this paper that ban information flow completely, there is also the need for ones that bound the capacity of any channel from high to low. Of course in a timed context we have the possibility of measuring this in bits per second.

Resources

FDR can be downloaded from http://www.cs.ox.ac.uk/projects/concurrency-tools/. Version 2.94 contains all the features used in this paper such as the Timed CSP implementation. Example files for FDR containing examples and case studies from this paper can be found together with the pre-print of this paper at http://www.cs.ox.ac.uk/people/publications/personal/Bill.Roscoe.html.

Notation of timed traces and failures

The following notation, used in this paper, is derived from the literature of Timed CSP and applies to the continuous model $\mathcal{F}_{\mathbf{T}}$.

(a, t) $(a \in \Sigma, t \in \mathbb{R}^+)$ a timed event
timed trace: a finite sequence of timed events $\langle (a_i, t_i) \mid i \in \langle 0 \ldots n-1 \rangle \rangle$ where $i < j \Rightarrow a_i \leq t_j$.
$X \times [t1, t2)$ $(X \subseteq \Sigma, 0 \leq t_1 < t_2 < \infty)$ a refusal token.
$\aleph$ a timed refusal: union of refusal tokens subject to only finitely many starting before any given t.
$(t, \aleph)$ timed failure: s a timed trace and $\aleph$ a timed refusal.
$s \upharpoonright t$ the sequence of all (timed) events in the trace s up to *and including* those at t.
$\aleph \upharpoonright t = \aleph \cap (\Sigma \times [0, t))$ the refusals in $\aleph$ up to and *not including* those at t.
$end(s)$ the last time appearing in s, or 0 if $s = \langle\rangle$.
$begin(s)$ the first time appearing in s, or ∞ if $s = \langle\rangle$.
$s^\frown t$ concatenation.

Acknowledgments

We are grateful to Joël Ouaknine for discussions on discrete Timed CSP and digitisation, to Phil Armstrong for implementing Timed CSP in FDR and to Long Nguyen for comments. This paper was greatly improved thanks to comments from anonymous referees. The work reported in this paper was partially supported by grants from EPSRC and ONR.

References

[All91] Allen PG (1991) A comparison of non-interference and non-deducibility using CSP. Proc CSFW. IEEE
[AlDi94] Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235
[AHR12] Armstrong P, Hopcroft PJ, Roscoe AW (2012) Fairness analysis through priority. Forthcoming
[ALOR12] Armstrong PJ, Lowe G, Ouaknine J, Roscoe AW (2012) Model-checking Timed CSP. Forthcoming HOWARD (H. Barringer festschift), Easychair (pub)
[BFST02] Barbuti R, Francesco ND, Santone A, Tesei L (2002) A notion of non-interference for timed automata. Fundam Inform 51:1–11
[BaTe03] Barbuti R, Tesei L (2003) A decidable notion of timed non-interference. Fundam Inform 54:137–150
[FoGo94] Focardi R, Gorrieri R (1994) A classification of security properties for process algebras. J Comput Secur 3:5–33
[FGM00] Focardi R, Gorrieri R, Martinelli F (2000) Information flow analysis in a discrete-time process algebra. CSFW-13, IEEE
[FGM03] Focardi R, Gorrieri R, Martinelli F (2003) Real-time information flow analysis. Sel Areas Commun 21:20–34
[For99] Forster R (1999) Noninterference properties for nondeterministic processes. Oxford University DPhil thesis
[FRR99] Forster R, Reed GM, Roscoe AW (2000) The successes and failures of behavioural models. In: Millenial perspectives in computer science. Palgrave
[GoMe82] Goguen JA, Meseguer J (1982) Security policies and security models. In: Proceedings of IEEE symposium on security and privacy
[GCu92] Graham-Cumming J (1992) The formal development of secure systems. Oxford University DPhil thesis
[HMP92] Henzinger TA, Manna Z, Pnueli A (1992) What good are digital clocks? In: Proceedings of the nineteenth international colloquium on automata, languages, and programming (ICALP 92), vol 623. Springer/LNCS, Berlin, pp 545–558
[Hua10] Huang J (2010) Extending non-interference properties to the timed world. Oxford University DPhil thesis
[HuRo06] Huang J, Roscoe AW (2006) Extending non-interference properties to the timed world. In: Proc ACM SAC
[Laz99] Lazić RS (1999) A semantic study of data independence with applications to model checking. Oxford University DPhil thesis
[LoOu06] Lowe G, Ouaknine J (2006) On timed models and full abstraction. ENTCS 155:497–519
[McIMo10] McIver AK, Morgan CC (2010) The thousand-and-one cryptographers. Reflections on the work of C.A.R. Hoare. Springer, Berlin
[Mor06] Morgan CC (2006) The shadow knows: refinement of ignorance in sequential programs. Proc MPC LNCS 4014
[Oua01] Ouaknine J (2001) Discrete analysis of continuous behaviour in real-time concurrent systems. Oxford University D.Phil thesis
[Oua02] Ouaknine J (2002) Digitisation and full abstraction for dense-time model checking. TACAS Springer LNCS
[OuWo03] Ouaknine J, Worrell JB (2003) Timed CSP = closed timed epsilon-automata. Nord J Comput 10:99–133
[Ree88] Reed GM (1988) A uniform mathematical theory for real-time distributed computing. Oxford University DPhil thesis
[ReRo88] Reed GM, Roscoe AW (1988) A timed model for communicating sequential processes. Theor Comput Sci 58:249–261
[ReRo99] Reed GM, Roscoe AW (1999) The timed failures-stability model for CSP. Theor Comput Sci 211:85–127
[Ros94] Roscoe AW (1994) Model checking CSP. In: A classical mind: essays in honour of C.A.R. Hoare. Prentice Hall

[Ros95] Roscoe AW (1995) CSP and determinism in security modelling. Proceedings of IEEE symposium on security and privacy
[Ros97] Roscoe AW (1997) The theory and practice of concurrency. Prentice Hall
[Ros06] Roscoe AW (2006) Confluence thanks to extensional determinism. ENTCS 162:305–309
[Ros10] Roscoe AW (2010) Understanding concurrent systems. Springer, Berlin
[RWW96] Roscoe AW, Woodcock JCP, Wulf L (1996) Non-interference through determinism. J Comput Secur 4(1):27–53
[Rya91] Ryan PYA (1991) A CSP formulation of non-interference and unwinding. Cipher Winter 1991. IEEE Press
[Sch00] Schneider SA (2000) Concurrent and real-time systems: the CSP approach. Wiley, New York

Received 18 January 2012
Accepted in revised form 25 May 2012 by Peter Höfner, Robert van Glabbeek and Ian Hayes
Published online 29 June 2012