On timed models and full abstraction

Gavin Lowe and Joël Ouaknine [1]

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK.

Abstract

In this paper we study a denotational model for a discrete-time version of CSP. We give a compositional semantics for the language. The model records refusal information at the end of each time unit; we believe this model to be simpler than existing models. We also show that the model is fully abstract: equivalence in the model corresponds to the natural equivalence of may testing; and all members of the denotational model are syntactically expressible. We also consider a slightly weaker model, containing no refusal information; we show that this model corresponds to an alternative form of may testing. We briefly discuss the application of these models to the study of information flow in multi-level secure systems.

Key words: Process algebra, timed models, full abstraction, CSP.

1 Introduction

In this paper, we consider denotational semantic models for a discrete-time version of CSP [2,10]. In particular, we present a compositional model, the *timed testing model*, for the language, which is fully abstract with respect to may testing, and hence is the coarsest reasonable model of the language. The original model for the language, the *refusal traces model* [2] from [6], represented a process by the set of its refusal traces, where a refusal trace is an alternating sequence of sets of events that are refused, and events that are performed by the process. The model of this paper records fewer refusal sets: it records refusals only before the special event *tock* that represents the passage of one time unit.

Our interest in this model arises from our work on information flow in multi-level secure systems. In a multi-level secure system, users are given se-

[1] Email: gavin.lowe@comlab.ox.ac.uk, joel.ouaknine@comlab.ox.ac.uk

[2] The model was originally called the *refusal testing model*; we use the name refusal traces, because we will be using the word "testing" in a different sense.

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

curity clearances (for example, “Top Secret”, “Secret”, “Confidential”, “Unclassified”). The question, then, is whether a user with a high classification level can pass information to a user with a lower classification level, and if so how much. In [4], we introduced a definition for the quantity of information that can be passed in such a system modelled in CSP. It turned out that the appropriate semantic model for this analysis was the timed testing model we consider in this paper. In that earlier paper, we defined that model as a projection of the refusal traces model from [6]. The current paper considers the timed testing model in more detail, and can be seen as building the foundations for the earlier paper.

Let’s look at an example. Suppose the system is willing to perform an event h from a high level user $High$, after which it will perform an event l of a low level user Low, and then return to its initial state after one time unit:

$$System = h \rightarrow l \rightarrow Wait\,1\,; System.$$

How can $High$ exploit this to pass information to Low? One tactic would be to perform h whenever he wants to pass a 1 bit, and to do nothing when he wants to pass a 0 bit; if we model the choice of bit by an input on channel in, this tactic is captured by the following process:

$$High = in?x \rightarrow \text{if } x = 1 \text{ then } h \rightarrow Wait\,1\,; High \text{ else } Wait\,1\,; High.$$

Low can then try to perform l; if this succeeds, then he knows $High$ was trying to pass 1; if this fails, he can timeout after one time unit and deduce that $High$ was trying to pass 0:

$$Low = (l \rightarrow Wait\,1\,; out!1 \rightarrow Low) \overset{1}{\rhd} (out!0 \rightarrow Low).$$

When we combine these processes in parallel, hiding the internal communications:

$$((High \underset{\{h\}}{\parallel} System) \underset{\{l\}}{\parallel} Low) \setminus \{h, l\}$$

the resulting system acts as a reliable buffer from in to out (albeit with a small delay). Hence this system can be used to pass one bit per time unit.

One can think of Low as a testing process, testing $High \underset{\{h\}}{\parallel} System$ in order to ascertain its behaviour. What is the appropriate model in which to reason about such systems? Clearly, we need some kind of refusal testing model [8,5] (i.e. where refusal information is recorded throughout a trace, rather than just at the end), for Low needs to be able to detect refusals of the system —corresponding to 0 bits in the example— and then continue testing. The inclusion of time in the model is necessary if we are to talk about flows of information caused by timing, or about the *rate* of information flow; it has the fortunate consequence of allowing observation of refusals to correspond to testing (as shown in Section 6). However, it turns out that the only refusals Low can detect —and that should be included in the model— are those just before a *tock* event: such refusals can be detected by employing a timeout, as in the above example.

In [4] we defined such a model as a projection of the timed refusal traces model (the model in this paper is isomorphic to, but slightly simpler than, the model in that paper), but did not study its properties. In this paper, we give a compositional semantics to discrete time CSP within this model. We then show that the model is fully abstract: two processes are distinguished in the model if and only if they are distinguished by may testing; and the model contains no junk, i.e. every element of the semantic domain corresponds to a syntactically-definable process.

In Section 2 we review the syntax of discrete-time CSP, and in Section 3 we describe the refusal traces model from [6]. In Section 4 we define the timed testing model, giving it a compositional semantics, and in Section 5 we consider the relationship between the two models, presenting a projection from the former to the latter. In Section 6 we prove the full abstraction result. Finally, in Section 7, we consider a model that records no refusal information: this model is not compositional, but can be defined as a projection of the timed testing model; we show that equivalence in this model corresponds to an alternative form of may testing. Some proofs are omitted because of lack of space; they can be found in the full version of the paper.

2 Timed CSP syntax

Let Σ be a finite set of *events*. We assume a special event *tock* (not in Σ) that represents the passage of one time unit. We define $\Sigma^{tock} = \Sigma \cup \{tock\}$.

In the notation below, we have $a \in \Sigma$ and $A \subseteq \Sigma$. The parameter n ranges over the non-negative integers $\mathbb{N}$. R represents a renaming relation $R : \Sigma \leftrightarrow \Sigma$. The variable X is drawn from a fixed infinite set of process variables VAR.

Timed CSP terms are constructed according to the following grammar:

$$\begin{aligned} P := {} & STOP \mid a \to P \mid Wait\, n\,; P \mid P_1 \overset{n}{\rhd} P_2 \mid P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid \\ & P_1 \underset{A}{\parallel} P_2 \mid P \setminus A \mid P[R] \mid X \mid \mu X \bullet P \ . \end{aligned}$$

These terms have the following intuitive interpretations:

- $STOP$ is the deadlocked, stable process that is only capable of letting time pass.
- $a \to P$ initially offers to engage in the event a at any time, and subsequently behaves like P.
- $Wait\, n\,; P$ is the process that idles for n time units, and then becomes P.
- $P \overset{n}{\rhd} Q$ is a timeout process that initially offers to act like P for n time units, after which it silently becomes Q if P has failed to communicate any visible event.
- $P \sqcap Q$ represents the nondeterministic (or internal) choice between P and Q; which of these two processes $P \sqcap Q$ becomes is independent of the environment: how the choice is resolved is considered to be outside the

domain of discourse.

- $P \;\Box\; Q$, on the other hand, denotes a process which is willing to behave either like P or Q, at the choice of the environment; this decision is taken on the first visible event that is communicated.
- $P_1 \underset{A}{\|} P_2$ is a parallel composition of P_1 and P_2, synchronising on all events in A.
- $P \setminus A$ is a process that behaves like P but with all communications in the set A hidden (made invisible to the environment); the assumption of *maximal progress*, or τ*-urgency*, dictates that no time can elapse while hidden events are on offer—in other words, hidden events happen as soon as they become available.
- $P[R]$ is a process that behaves like P except all the (non-*tock*) events are renamed according to the relation R; so if P can perform a, then $P[R]$ can perform any b such that aRb; note that the renaming leaves *tock*s unchanged[3] .
- The process variable X has no intrinsic behaviour of its own, but can imitate any process P as part of a recursion—see below.
- The recursion $\mu X \bullet P$ represents a process which behaves like P but with every free occurrence of X in P (recursively) replaced by $\mu X \bullet P$; semantically, this corresponds to the unique solution to the equation $X = P$; note that the variable X here usually appears freely within P's body.

Note our requirement that all *delays* (parameters n in the terms $Wait\, n$ and $P_1 \overset{n}{\rhd} P_2$) be integral. This restriction is essentially harmless, because of the freedom to scale time units. We could equivalently have required *rational* delays, as many authors do.

Within the recursive process $\mu X \bullet P$, we specify that X should be *time-guarded* in P, i.e. the process cannot recurse without any time passing. This condition prevents Zeno processes that do not allow time to progress. Sufficient syntactical conditions for time-guardedness are given in [6].

We introduce some derived constructs. We write $\bigsqcap_{i:I} P_i$ and $\Box_{i:I} P_i$ for indexed internal and external choices, respectively, indexed by the finite set I. We write $?a : A \to P(a)$ for $\Box_{a:A}\, a \to P(a)$, i.e. a process that initially offers the events of A, and when one of them is performed acts like the corresponding $P(a)$. We write $P \;|||\; Q$ for an interleaving of P and Q, i.e. $P \underset{\{\}}{\|} Q$. We also tend to express recursions by means of the equational notation $X = P$, rather than the functional $\mu X \bullet P$ prescribed by the definition.

We use the conventions that prefixing ($\to$) and sequential composition (;) bind tighter than the binary choice operators ($\sqcap$, $\Box$, $\rhd$).

[3] This construct generalises the functional renaming ($f(P)$) and inverse renaming ($f^{-1}(P)$) constructs found in early versions of CSP.

Note that we change the semantics of the timeout operator slightly from that of [6,7]: our timeouts are *strict*, in the sense that events of P cannot be performed by $P \overset{n}{\rhd} Q$ after the nth *tock*; by contrast, in [6,7], events of P are still available (unstably) after the nth, but before the $n+1$th, *tock*. This non-strict timeout, written as "$\overset{n}{\triangleright}$", can be defined as a derived operator, as follows:

$$P \overset{n}{\triangleright} Q = (P \;\Box\; Wait\, n \,;\, trig \rightarrow Q) \setminus \{trig\},$$

where $trig$ is a fresh event. (By contrast, the strict timeout cannot be defined as a derived operator.) We originally introduced the strict timeout because it seems more appropriate for modelling purposes: for example, it is necessary in the information flow example from the introduction. Moreover, it also seems to be necessary for the "no junk" result in Section 6.

3 The timed refusal traces model

The timed refusal traces model, in common with most semantic models of CSP, contains information about both which events are performed, and which events are refused by the process. Recall that a set of events X is said to be *refused* by a process P if P is stable, i.e. is unable to perform any internal events, and is unable to perform any of the events from X. The timed refusal trace model for CSP records refusal information about a process throughout a trace (by contrast, the untimed stable failures model [10] records refusal information only at the end of the trace).

However, it might be the case that no refusal information can be recorded at a particular point in a trace, because the process does not enter a stable state. For example, consider the process $(a \rightarrow STOP \;\Box\; b \rightarrow STOP) \setminus \{a\}$. If this process performs a b, then it does so from an unstable state, where the internal event a is available; hence no refusal can be observed before the b. The model deals with this situation by using a *null refusal*, written as $\bullet$, to indicate that no refusal information was recorded, possibly because the process did not enter a stable state; for example, the above process has the trace $\langle \bullet, b, \{a, b, c\}\rangle$.

Formally, we define the set REF of refusal information by

$$REF \mathrel{\widehat{=}} \mathbb{P}\,\Sigma \cup \{\bullet\}.$$

We lift the normal set operators to REF in the obvious way (decorating the lifted versions with "$*$"), treating $\bullet$ as being below the empty set; for example, for $x \in \Sigma$, $X \in \mathbb{P}\,\Sigma$, we have $\bullet \subseteq^* X$; $\bullet \cup^* X = \bullet \cap^* X = \bullet$; $\neg(x \in^* \bullet)$.

The timed refusal trace model for CSP represents a process by its refusal traces, i.e. alternating sequences of the form $\langle X_0, a_1, X_1, a_2, \ldots, a_n, X_n\rangle$, where each $a_i \in \Sigma^{tock}$ represents the performance of the event a_i, and each $X_i \in REF$ represents refusal information. Formally, we define refusal traces using the regular expression

$$RefusalTrace \mathrel{\widehat{=}} REF.(\Sigma^{tock}.REF)^*.$$

We define some operations over refusal traces. $trace(tr)$ removes all refusal information from tr:

$$\begin{aligned} trace(\langle\rangle) &\mathrel{\widehat{=}} \langle\rangle, \\ trace(\langle a\rangle^\frown tr) &\mathrel{\widehat{=}} \langle a\rangle^\frown trace(tr), && \text{if } a \in \Sigma^{tock}, \\ trace(\langle X\rangle^\frown tr) &\mathrel{\widehat{=}} trace(tr), && \text{if } X \in REF. \end{aligned}$$

$refusals(tr)$ returns the set of events that are refused during tr [4]:

$$refusals(tr) \mathrel{\widehat{=}} \bigcup\{X \in \mathbb{P}\,\Sigma \mid X \text{ in } tr\}.$$

$tr \preceq tr'$ indicates that tr contains less information than tr' in the sense that tr is a prefix of tr', or it contains smaller refusal sets, or a combination of the two. Formally, $\preceq$ is the smallest relation satisfying

$$\begin{aligned} \langle X\rangle \preceq \langle Y\rangle^\frown tr &\Leftarrow X \subseteq^* Y, \\ \langle X, a\rangle^\frown tr \preceq \langle Y, a\rangle^\frown tr' &\Leftarrow X \subseteq^* Y \wedge tr \preceq tr'. \end{aligned}$$

The refusal traces model $\mathcal{M}_{\mathcal{R}}$ contains those $S \subseteq RefusalTrace$ such that the following conditions hold, where a ranges over Σ and A over REF [5].

R1. $\langle\bullet\rangle \in S$;

R2. $tr \in S \wedge tr' \preceq tr \Rightarrow tr' \in S$;

R3. $tr^\frown\langle\bullet, tock\rangle^\frown tr' \in S \Rightarrow tr^\frown\langle\{\}, tock\rangle^\frown tr' \in S$;

R4. $A \neq \bullet \wedge tr^\frown\langle A\rangle^\frown tr' \in S \wedge tr^\frown\langle A, a, \bullet\rangle \notin S \Rightarrow tr^\frown\langle A \cup \{a\}\rangle^\frown tr' \in S$;

R5. $tr \in S \Rightarrow tr^\frown\langle tock, \bullet\rangle \in S$;

R6. $tr^\frown\langle A, a\rangle^\frown tr' \in S \Rightarrow a \notin^* A$;

R7. $\forall k \in \mathbb{N} \bullet \exists n \in \mathbb{N} \bullet \forall tr \in S \bullet \#(tr \upharpoonright tock) \leq k \Rightarrow \#(trace(tr)) \leq n$.

The first condition says that the minimal behaviour $\langle\bullet\rangle$ can always be observed. The second condition says that the observations of S are downwards-closed. Condition R3 says that a process will stabilise before a *tock*, so at least the empty set can be refused. Condition R4 says that if a process can stably refuse A, and cannot perform a, then a can be added to the refusal set. Condition R5 says that time cannot be stopped. Condition R6 says that an event a cannot be refused and then performed. Condition R7 is a bounded-speed condition: there is a bound n on the number of events that can be performed in the first k time units. [6]

[4] The notation x in tr indicates that x is an element of the sequence, or trace, tr.

[5] The operator $\upharpoonright$ restricts a trace to a particular event or set of events.

[6] In [6], there was an extra condition: $A \neq \bullet \wedge tr^\frown\langle A, a, \bullet\rangle \in S \Rightarrow tr^\frown\langle A, tock, \bullet, a, \bullet\rangle \in S$, which says that if an event is stably available, then it will still be available after a *tock*. This condition is not satisfied by our strict timeout operator. Further, it seems less natural to us than the other conditions.

3.1 Semantic definitions

We can define a semantic function

$$\mathcal{R} : CSP \to (VAR \to \mathcal{M}_\mathcal{R}) \to \mathcal{M}_\mathcal{R},$$

such that $\mathcal{R}[\![P]\!]\rho$ gives the refusal traces of P, assuming environment ρ gives the semantics for free variables of P. Throughout the following definitions, tr ranges over $RefusalTrace$, and $\widehat{tr}$ over prefixes of elements of $RefusalTrace$ ending in an event.

$$\begin{aligned}
&\mathcal{R}[\![STOP]\!]\rho \mathrel{\widehat{=}} \{tr \mid trace(tr) \in tock^*\},\\
&\mathcal{R}[\![a \to P]\!]\rho \mathrel{\widehat{=}}\\
&\quad \{tr \mid trace(tr) \in tock^* \wedge a \notin refusals(tr)\} \cup\\
&\quad \{tr^\frown\langle a\rangle^\frown tr' \mid trace(tr) \in tock^* \wedge a \notin refusals(tr) \wedge tr' \in \mathcal{R}[\![P]\!]\rho\},\\
&\mathcal{R}[\![Wait\, n\,;P]\!]\rho \mathrel{\widehat{=}}\\
&\quad \{tr \mid trace(tr) < tock^n\} \cup\\
&\quad \{\widehat{tr}^\frown tr' \mid trace(\widehat{tr}) = tock^n \wedge tr' \in \mathcal{R}[\![P]\!]\rho\},\\
&\mathcal{R}[\![P \,\square\, Q]\!]\rho \mathrel{\widehat{=}}\\
&\quad \{tr \mid trace(tr) \in tock^* \wedge tr \in \mathcal{R}[\![P]\!]\rho \cap \mathcal{R}[\![Q]\!]\rho\} \cup\\
&\quad \{tr^\frown\langle a\rangle^\frown tr' \mid trace(tr) \in tock^* \wedge tr \in \mathcal{R}[\![P]\!]\rho \cap \mathcal{R}[\![Q]\!]\rho \wedge\\
&\qquad\qquad\qquad a \in \Sigma \wedge tr^\frown\langle a\rangle^\frown tr' \in \mathcal{R}[\![P]\!]\rho \cup \mathcal{R}[\![Q]\!]\rho\},\\
&\mathcal{R}[\![P \sqcap Q]\!]\rho \mathrel{\widehat{=}} \mathcal{R}[\![P]\!]\rho \cup \mathcal{R}[\![Q]\!]\rho,\\
&\mathcal{R}[\![P \overset{n}{\rhd} Q]\!]\rho \mathrel{\widehat{=}}\\
&\quad \{tr \mid tr \in \mathcal{R}[\![P]\!]\rho \wedge tock^n \not\leq trace(tr)\} \cup\\
&\quad \{\widehat{tr}^\frown tr' \mid \widehat{tr}^\frown\langle\bullet\rangle \in \mathcal{R}[\![P]\!]\rho \wedge trace(\widehat{tr}) = tock^n \wedge tr' \in \mathcal{R}[\![Q]\!]\rho\},\\
&\mathcal{R}[\![P \underset{A}{\|} Q]\!]\rho \mathrel{\widehat{=}} \{tr \mid \exists\, tr_P \in \mathcal{R}[\![P]\!]\rho, tr_Q \in \mathcal{R}[\![Q]\!]\rho \bullet tr \in tr_P \underset{A}{\|} tr_Q\},\\
&\mathcal{R}[\![P \setminus A]\!]\rho \mathrel{\widehat{=}} \mathsf{RefCl}\{tr \setminus A \mid tr \in \mathcal{R}[\![P]\!]\rho \wedge tr \text{ is } A\text{-urgent}\},\\
&\mathcal{R}[\![P[R]]\!]\rho \mathrel{\widehat{=}} \{tr' \mid \exists\, tr \in \mathcal{R}[\![P]\!]\rho \bullet tr\, R^*\, tr'\},\\
&\mathcal{R}[\![X]\!]\rho \mathrel{\widehat{=}} \rho(X),\\
&\mathcal{R}[\![\mu\, X \bullet P]\!]\rho \mathrel{\widehat{=}} \text{the unique fixed point of the mapping } C : \mathcal{M}_\mathcal{R} \to \mathcal{M}_\mathcal{R}\\
&\qquad\qquad\qquad \text{given by } C(S) \mathrel{\widehat{=}} \mathcal{R}[\![P]\!]\rho[S/X].
\end{aligned}$$

The semantics of parallel composition, hiding and renaming use some auxiliary functions. $tr_P \underset{A}{\|} tr_Q$ represents the set of traces that can be formed from the parallel composition of traces tr_P and tr_Q, synchronising on events from $A^{tock} \mathrel{\widehat{=}} A \cup \{tock\}$; this operator is defined by the equations in Figure 1 and the obvious symmetric equations. Within that definition, if the two processes refuse X and Y respectively, then the composition can refuse any set from

$$X \underset{A}{\cap} Y \mathrel{\widehat{=}} \{C \mid C \subseteq^* (X \cap^* A) \cup^* (Y \cap^* A) \cup^* (X \cap^* Y)\}.$$

$$
\begin{aligned}
&\langle X\rangle \underset{A}{\|} \langle Y\rangle \mathrel{\widehat{=}} \{\langle C\rangle \mid C \in X \underset{A}{\cap} Y\},\\
&\langle X,x\rangle^\frown tr \underset{A}{\|} \langle Y\rangle \mathrel{\widehat{=}}\\
&\quad \text{if } x \in A^{tock} \text{ then } \{\} \text{ else } \{\langle C,x\rangle^\frown tr' \mid C \in X \underset{A}{\cap} Y \wedge tr' \in tr \underset{A}{\|} \langle Y\rangle\},\\
&\langle X,x\rangle^\frown tr \underset{A}{\|} \langle Y,y\rangle^\frown tr' \mathrel{\widehat{=}}\\
&\quad \text{if } x \notin A^{tock} \wedge y \in A^{tock}\\
&\quad \text{then } \{\langle C,x\rangle^\frown tr'' \mid C \in X \underset{A}{\cap} Y \wedge tr'' \in tr \underset{A}{\|} \langle Y,y\rangle^\frown tr'\}\\
&\quad \text{else if } x \in A^{tock} \wedge y \notin A^{tock}\\
&\quad \text{then } \{\langle C,y\rangle^\frown tr'' \mid C \in X \underset{A}{\cap} Y \wedge tr'' \in \langle X,x\rangle^\frown tr \underset{A}{\|} tr'\}\\
&\quad \text{else if } x \notin A^{tock} \wedge y \notin A^{tock}\\
&\quad \text{then } \{\langle C,x\rangle^\frown tr'' \mid C \in X \underset{A}{\cap} Y \wedge tr'' \in tr \underset{A}{\|} \langle Y,y\rangle^\frown tr'\} \cup\\
&\qquad\qquad \{\langle C,y\rangle^\frown tr'' \mid C \in X \underset{A}{\cap} Y \wedge tr'' \in \langle X,x\rangle^\frown tr \underset{A}{\|} tr'\}\\
&\quad \text{else if } x = y \in A^{tock}\\
&\quad \text{then } \{\langle C,x\rangle^\frown tr'' \mid C \in X \underset{A}{\cap} Y \wedge tr'' \in tr \underset{A}{\|} tr'\} \text{ else } \{\}.
\end{aligned}
$$

Fig. 1. Parallel composition of traces in the refusal traces model

$$
\begin{aligned}
&tr \text{ is } A\text{-urgent} \Leftrightarrow\\
&\quad \forall X \in REF\,; tr', tr'' \mid tr = tr'^\frown\langle X, tock\rangle^\frown tr'' \bullet A \subseteq^* X.\\
&\langle X\rangle \setminus A \mathrel{\widehat{=}} \text{if } A \subseteq^* X \text{ then } \langle X\rangle \text{ else } \langle\bullet\rangle,\\
&(\langle X,x\rangle^\frown tr) \setminus A \mathrel{\widehat{=}} \text{if } x \in A \text{ then } tr \setminus A \text{ else } (\langle X\rangle \setminus A)^\frown\langle x\rangle^\frown(tr \setminus A).\\
&\mathsf{RefCl}\, S \mathrel{\widehat{=}} \{tr \mid \exists\, tr' \in S \bullet tr \preceq tr'\}.\\
&X\,\widehat{R}\,Y \Leftrightarrow X = Y = \bullet \vee X = \{x \mid \exists\, y \in Y \bullet xRy\}.\\
&\langle X\rangle\, R^*\, \langle Y\rangle \Leftrightarrow X\,\widehat{R}\,Y,\\
&(\langle X, tock\rangle^\frown tr)\, R^*\, (\langle Y, tock\rangle^\frown tr') \Leftrightarrow X\,\widehat{R}\,Y \wedge tr\, R^*\, tr',\\
&(\langle X, x\rangle^\frown tr)\, R^*\, (\langle Y, y\rangle^\frown tr') \Leftrightarrow X\,\widehat{R}\,Y \wedge xRy \wedge tr\, R^*\, tr', \quad \text{for } x, y \in \Sigma.
\end{aligned}
$$

Fig. 2. Auxiliary semantic definitions for hiding and renaming

Auxiliary semantic definitions relating to the definitions for $P \setminus A$ and $P[R]$ are in Figure 2. The assumption about the urgency of hidden events is captured by the fact that we consider only A-urgent traces of P: traces where all of A can be refused before a *tock*. $tr \setminus A$ represents the effect of hiding A in tr. $\mathsf{RefCl}\, S$ forms the downwards closure of S. For example, if

$$P = a \to (b \to STOP \;\Box\; a \to STOP),$$

then $P \setminus \{a\}$ has the refusal trace $\langle \bullet, b, \{b\}, tock, \bullet\rangle$, corresponding to the refusal trace $\langle \{b\}, a, \{c\}, b, \{a,b\}, tock, \bullet\rangle$ (among others) of P; note that $\{a\}$-urgency is necessary to ensure that a *tock* cannot happen before the first a or the b. $\widehat{R}$ represents the lifting of the renaming relation R to REF, and R^* represents the lifting to refusal traces.

In [6] it is shown that this denotational semantics is congruent to the operational semantics. The following theorem is adapted from [6].

Theorem 3.1 *The mapping $\mathcal{R}$ is well defined.*

4 The timed testing model

In this section we present the timed testing model for discrete-time CSP. The model records refusal information only before *tock*s. Further, we can do away with the null refusals $\bullet$ (essentially because of condition R3) and only record actual refusals from $\mathbb{P}\,\Sigma$. Timed tests can be defined using the following regular expression.

$$TimedTest \mathrel{\widehat{=}} (\Sigma + \mathbb{P}\,\Sigma.tock)^*.$$

Note that refusals are recorded before, and only before, *tock*s; for example $\langle a, b, \{b, c\}, tock, \{\}, tock, c\rangle$.

Note that this model fails to distinguish processes that are distinguished in the refusal traces model. For example, consider:

$$P \mathrel{\widehat{=}} ((a \to STOP \overset{1}{\rhd} STOP) \sqcap STOP),$$
$$Q \mathrel{\widehat{=}} (a \to STOP \mathbin{\Box} b \to STOP) \setminus \{b\}.$$

These are distinguished in the refusal traces model, for P has the refusal trace $\langle \{\}, a, \bullet\rangle$, which Q does not, for Q performs a from an unstable state. However, P and Q are equivalent in this model, for they can both perform an a only before the first *tock*, and can refuse anything before a *tock*. In Section 5 we show that the refusal traces model distinguishes all processes that are distinguished by this model, so this model is strictly more abstract.

We adapt some operations from refusal traces to timed tests. $trace(tr)$ removes all refusal information from tr. $refusals(tr)$ gives the set of all events that are refused anywhere in tr. These two operations are defined analogously to as in the refusal trace model. $tr \preceq tr'$ indicates that tr contains less information than tr':

$$\begin{aligned}
\langle\rangle &\preceq tr,\\
\langle a\rangle^\frown tr \preceq \langle a\rangle^\frown tr' &\Leftarrow tr \preceq tr',\\
\langle X, tock\rangle^\frown tr \preceq \langle Y, tock\rangle^\frown tr' &\Leftarrow X \subseteq Y \wedge tr \preceq tr'.
\end{aligned}$$

The timed testing model $\mathcal{M}_{\mathcal{T}}$ contains those $S \subseteq TimedTest$ such that the following conditions hold:

T1. $\langle\rangle \in S$;

T2. $tr \in S \wedge tr' \preceq tr \Rightarrow tr' \in S$;

T3. $tr^\frown\langle A, tock\rangle^\frown tr' \in S \wedge tr^\frown\langle a\rangle \notin S \Rightarrow tr^\frown\langle A \cup \{a\}, tock\rangle^\frown tr' \in S$;

T4. $tr \in S \Rightarrow tr^\frown\langle \{\}, tock\rangle \in S$;

T5. $\forall k \in \mathbb{N} \bullet \exists n \in \mathbb{N} \bullet \forall tr \in S \bullet \#(tr \restriction tock) \leq k \Rightarrow \#(trace(tr)) \leq n$.

These conditions correspond closely to conditions R1, R2, R4, R5 and R7, respectively.

4.1 Semantic definitions

We can define a semantic function

$$\mathcal{T} : CSP \to (VAR \to \mathcal{M}_{\mathcal{T}}) \to \mathcal{M}_{\mathcal{T}},$$

such that $\mathcal{T}[\![P]\!]\rho$ gives the timed tests of P, assuming environment ρ gives the semantics for free variables of P. Throughout these equations, tr ranges over *TimedTest*.

$$\mathcal{T}[\![STOP]\!]\rho \mathrel{\widehat{=}} \{tr \mid trace(tr) \in tock^*\},$$

$$\begin{aligned}\mathcal{T}[\![a \to P]\!]\rho \mathrel{\widehat{=}}& \\ &\{tr \mid trace(tr) \in tock^* \wedge a \notin refusals(tr)\} \cup \\ &\{tr ^\frown \langle a \rangle ^\frown tr' \mid trace(tr) \in tock^* \wedge a \notin refusals(tr) \wedge tr' \in \mathcal{T}[\![P]\!]\rho\},\end{aligned}$$

$$\begin{aligned}\mathcal{T}[\![Wait\, n\, ; P]\!]\rho \mathrel{\widehat{=}}& \\ &\{tr \mid trace(tr) < tock^n\} \cup \\ &\{tr ^\frown tr' \mid trace(tr) = tock^n \wedge tr' \in \mathcal{T}[\![P]\!]\rho\},\end{aligned}$$

$$\begin{aligned}\mathcal{T}[\![P \mathrel{\Box} Q]\!]\rho \mathrel{\widehat{=}}& \\ &\{tr \mid trace(tr) \in tock^* \wedge tr \in \mathcal{T}[\![P]\!]\rho \cap \mathcal{T}[\![Q]\!]\rho\} \cup \\ &\{tr ^\frown \langle a \rangle ^\frown tr' \mid trace(tr) \in tock^* \wedge tr \in \mathcal{T}[\![P]\!]\rho \cap \mathcal{T}[\![Q]\!]\rho \wedge \\ &\qquad\qquad a \in \Sigma \wedge tr ^\frown \langle a \rangle ^\frown tr' \in \mathcal{T}[\![P]\!]\rho \cup \mathcal{T}[\![Q]\!]\rho\},\end{aligned}$$

$$\mathcal{T}[\![P \sqcap Q]\!]\rho \mathrel{\widehat{=}} \mathcal{T}[\![P]\!]\rho \cup \mathcal{T}[\![Q]\!]\rho,$$

$$\begin{aligned}\mathcal{T}[\![P \overset{n}{\rhd} Q]\!]\rho \mathrel{\widehat{=}}& \\ &\{tr \mid tr \in \mathcal{T}[\![P]\!]\rho \wedge tock^n \not\leq trace(tr)\} \cup \\ &\{tr ^\frown tr' \mid tr \in \mathcal{T}[\![P]\!]\rho \wedge trace(tr) = tock^n \wedge tr' \in \mathcal{T}[\![Q]\!]\rho\},\end{aligned}$$

$$\mathcal{T}[\![P \underset{A}{\parallel} Q]\!]\rho \mathrel{\widehat{=}} \{tr \mid \exists tr_P \in \mathcal{T}[\![P]\!]\rho, tr_Q \in \mathcal{T}[\![Q]\!]\rho \bullet tr \in tr_P \underset{A}{\parallel} tr_Q\},$$

$$\mathcal{T}[\![P \setminus A]\!]\rho \mathrel{\widehat{=}} \mathsf{RefCl}\{tr \setminus A \mid tr \in \mathcal{T}[\![P]\!]\rho \wedge tr \text{ is } A\text{-urgent}\},$$

$$\mathcal{T}[\![P[R]]\!]\rho \mathrel{\widehat{=}} \{tr' \mid \exists tr \in \mathcal{T}[\![P]\!]\rho \bullet tr\, R^*\, tr'\},$$

$$\mathcal{T}[\![X]\!]\rho \mathrel{\widehat{=}} \rho(X),$$

$$\begin{aligned}\mathcal{T}[\![\mu X \bullet P]\!]\rho \mathrel{\widehat{=}}& \text{ the unique fixed point of the mapping } C : \mathcal{M}_{\mathcal{T}} \to \mathcal{M}_{\mathcal{T}} \\ &\text{ given by } C(S) \mathrel{\widehat{=}} \mathcal{T}[\![P]\!]\rho[S/X].\end{aligned}$$

The auxilliary operators relating to parallel composition, hiding and renaming are similar to as in Figures 1 and 2.

Theorem 4.1 *The mapping $\mathcal{T}$ is well defined.*

Proof. We need to show that if $\rho \in VAR \to \mathcal{M}_{\mathcal{T}}$ then $\mathcal{T}[\![P]\!]\rho \in \mathcal{M}_{\mathcal{T}}$, i.e. that $\mathcal{T}[\![P]\!]\rho$ satisfies the healthiness conditions of the semantic model. This is a large structural induction; most cases are straightforward. As an illustrative example, we include the proof of T4 for hiding.

Suppose $tr \in \mathcal{T}[\![P \setminus A]\!]\rho$. Then there is some $tr' \in \mathcal{T}[\![P]\!]\rho$ such that tr' is A-urgent, and $tr \in \mathsf{RefCl}\{tr' \setminus A\}$. We show that, after tr', for every n, either n events from A can be performed, or all of A can be refused after $k < n$ events:

$$\forall n \in \mathbb{N} \bullet \exists a_1, \ldots, a_n \in A \bullet tr'^\frown\langle a_1, \ldots, a_n\rangle \in \mathcal{T}[\![P]\!]\rho \vee \\ \exists k < n \bullet tr'^\frown\langle a_1, \ldots, a_k, A, tock\rangle \in \mathcal{T}[\![P]\!]\rho.$$

The proof is by induction on n. The base case is immediate. For the inductive case, suppose $tr'^\frown\langle a_1, \ldots, a_n\rangle \in \mathcal{T}[\![P]\!]\rho$. If there is some $a_{n+1} \in A$ such that $tr'^\frown\langle a_1, \ldots, a_{n+1}\rangle \in \mathcal{T}[\![P]\!]\rho$ then we are done. Otherwise, by T4 we have that $tr'^\frown\langle a_1, \ldots, a_n, \{\}, tock\rangle \in \mathcal{T}[\![P]\!]\rho$. Then by repeated use of T3 we have that $tr'^\frown\langle a_1, \ldots, a_n, A, tock\rangle \in \mathcal{T}[\![P]\!]\rho$, as required.

Because of condition T5, there is a bound on the number of events that can be added to the trace using the process in the previous paragraph. Hence we have that $tr'' \mathrel{\widehat{=}} tr'^\frown\langle a_1, \ldots, a_k, A, tock\rangle \in \mathcal{T}[\![P]\!]\rho$ for some $a_1, \ldots, a_k \in A$. Now, tr'' is A-urgent, and $tr^\frown\langle\{\}, tock\rangle \in \mathsf{RefCl}\{tr'' \setminus A\}$ so $tr^\frown\langle\{\}, tock\rangle \in \mathcal{T}[\![P \setminus A]\!]\rho$, as required.

The case of recursion is almost identical to as in [6] for the refusal testing model. An ultra metric is defined over $\mathcal{M}_\mathcal{T}$ as follows. Given $S \in \mathcal{M}_\mathcal{T}$, write $S(t)$ for those refusal traces in S containing fewer than t tocks:

$$S(t) \mathrel{\widehat{=}} \{tr \in S \mid \#(trace(tr) \upharpoonright tock) < t\}.$$

(In particular, $S(0) = \{\}$.) Informally, if two processes behave the same before time t, but then behave differently, then they are at distance 2^{-t}; more precisely:

$$d(S, S') \mathrel{\widehat{=}} \inf\{2^{-t} \mid S(t) = S'(t)\}.$$

One can then show that $\mathcal{M}_\mathcal{T}$ is a complete metric space with respect to this metric. If P is time-guarded in X, then the corresponding mapping $C(S) \mathrel{\widehat{=}} \mathcal{T}[\![P]\!]\rho[S/X]$ is a contraction mapping, i.e.:

$$d(C(S), C(S')) \leq \frac{1}{2} d(S, S').$$

One can then use Banach's fixed point theorem (see e.g. [13]) to show that C has a unique fixed point, and further that fixed point equals $\lim_{n\to\infty} C^n(Q)$ for an arbitrary process Q. □

Note that the existence of a compositional semantic function implies that equivalence in this model is a congruence with respect to the CSP operators.

5 Relating the two models

In this section we consider the relationship between the refusal trace and timed testing models. Recall that there are two differences between the two models:

(1) the timed testing model contains refusal information only before *tock*s; and (2) the timed testing model does not use the null refusal $\bullet$. We therefore define a function

$$f : \mathit{RefusalTrace} \to \mathit{TimedTest}$$

that maps a refusal trace by the corresponding timed test by (1) removing all refusal information except that preceding *tock*s; and (2) replacing any occurrence of $\bullet$ before *tock* by $\{\}$:

$$\begin{aligned} f(\langle A\rangle) &\mathrel{\widehat{=}} \langle\rangle, \\ f(\langle A, a\rangle^\frown tr) &\mathrel{\widehat{=}} \langle a\rangle^\frown f(tr), \quad \text{for } a \in \Sigma, \\ f(\langle \bullet, tock\rangle^\frown tr) &\mathrel{\widehat{=}} \langle \{\}, tock\rangle^\frown f(tr), \\ f(\langle A, tock\rangle^\frown tr) &\mathrel{\widehat{=}} \langle A, tock\rangle^\frown f(tr), \quad \text{for } A \in \mathbb{P}\,\Sigma. \end{aligned}$$

We then lift f to sets of refusal traces by pointwise application:

$$f(S) \mathrel{\widehat{=}} \{f(tr) \mid tr \in S\}.$$

We begin by showing that f maps elements of $\mathcal{M}_{\mathcal{R}}$ into $\mathcal{M}_{\mathcal{T}}$:

Theorem 5.1 *If $S \in \mathcal{M}_{\mathcal{R}}$ then $f(S) \in \mathcal{M}_{\mathcal{T}}$.*

Proof. (sketch) That $f(S)$ satisfies the axioms of $\mathcal{M}_{\mathcal{T}}$ can be easily shown from the fact that S satisfies the corresponding axioms of $\mathcal{M}_{\mathcal{R}}$. □

We now show that f maps any syntactic process in $\mathcal{M}_{\mathcal{R}}$ to the corresponding process in $\mathcal{M}_{\mathcal{T}}$, i.e. f forms a homomorphism with respect to the CSP operators.

Theorem 5.2 *If $\forall X \bullet \rho_T(X) = f(\rho_R(X))$ then $\mathcal{T}[\![P]\!]\rho_T = f(\mathcal{R}[\![P]\!]\rho_R)$.*

Proof. (sketch). This is a large structural induction over the syntax of the language. Some cases can be simplified as follows. Define the function

$$g : \mathit{TimedTest} \to \mathit{RefusalTrace}$$

to insert $\bullet$ before every element of Σ and at the end of the trace:

$$\begin{aligned} g(\langle\rangle) &\mathrel{\widehat{=}} \langle\bullet\rangle, \\ g(\langle a\rangle^\frown tr) &\mathrel{\widehat{=}} \langle \bullet, a\rangle^\frown g(tr), \quad \text{for } a \in \Sigma, \\ g(\langle A, tock\rangle^\frown tr) &\mathrel{\widehat{=}} \langle A, tock\rangle^\frown g(tr). \end{aligned}$$

Note that $f(g(tr)) = tr$. Then the result of the theorem is equivalent to the following two statements:

$$\begin{aligned} &\forall\, tr \in \mathit{RefusalTrace} \bullet tr \in \mathcal{R}[\![P]\!]\rho_R \Rightarrow f(tr) \in \mathcal{T}[\![P]\!]\rho_T, \\ &\forall\, tr \in \mathit{TimedTest} \bullet tr \in \mathcal{T}[\![P]\!]\rho_T \Rightarrow g(tr) \in \mathcal{R}[\![P]\!]\rho_R. \end{aligned}$$

Proving these two statements, inductively, is straightforward in most cases.

For recursion, consider the recursive process $\mu\, X \bullet P$. Let $C_T : \mathcal{M}_{\mathcal{T}} \to \mathcal{M}_{\mathcal{T}}$ and $C_R : \mathcal{M}_{\mathcal{R}} \to \mathcal{M}_{\mathcal{R}}$ be the corresponding mappings on the semantic models, i.e. $C_T(S) \mathrel{\widehat{=}} \mathcal{T}[\![P]\!]\rho_T[S/X]$, and $C_R(S) \mathrel{\widehat{=}} \mathcal{R}[\![P]\!]\rho_R[S/X]$. Our structural induction hypothesis shows that

$$C_T(f(S)) = \mathcal{T}[\![P]\!]\rho_T[f(S)/X] = f(\mathcal{R}[\![P]\!]\rho_R[S/X]) = f(C_R(S)).$$

As shown in the fixed point theorems, the fixed points are the limits of the sequences $(R_n)_{n\in\mathbb{N}}$ and $(T_n)_{n\in\mathbb{N}}$, where

$$\begin{array}{ll} T_0 \mathrel{\widehat{=}} \mathcal{T}[\![STOP]\!]\rho_T, & T_{n+1} \mathrel{\widehat{=}} C_T(T_n), \\ R_0 \mathrel{\widehat{=}} \mathcal{R}[\![STOP]\!]\rho_R, & R_{n+1} \mathrel{\widehat{=}} C_R(R_n). \end{array}$$

We show that $T_n = f(R_n)$ by induction on n. The base case is trivial. For the inductive case:

$$T_{n+1} = C_T(T_n) = C_T(f(R_n)) = f(C_R(R_n)) = f(R_{n+1}),$$

using the inductive hypothesis and the above result. Hence

$$\begin{aligned} \mathcal{T}[\![\, \mu\, X \bullet P]\!]\rho_T &= \lim_{n\to\infty} T_n = \lim_{n\to\infty} f(R_n) \\ &= f(\lim_{n\to\infty} R_n) = f(\mathcal{R}[\![\, \mu\, X \bullet P]\!]\rho_R), \end{aligned}$$

using the continuity of f. □

6 Full abstraction

A semantic model is said to be *fully abstract* if the following two conditions hold [10]:

- Whenever two processes P and Q are distinguished by the semantics, then they are distinguished by some natural criterion: typically this criterion is the existence of a context $C[\_]$ such that one of $C[P]$ and $C[Q]$ passes, and the other fails, some simple test.
- Every element of the semantic model corresponds to a syntactically expressible process (taking a liberal view of expressibility, for example allowing mutual recursions to be indexed by non-constructive mathematical objects); this condition is sometimes referred to as “no junk”.

(The no junk condition is not always considered to be part of full abstraction.) In the next subsection we prove that the timed testing model satisfies the former condition by showing that equivalence in the model corresponds to may testing equivalence [1]. In the following subsection we show that the model satisfies the no junk condition.

6.1 Timed may testing

A *timed test process* is simply a timed CSP process that can perform a special event $\omega \notin \Sigma$ representing success. When defining test processes, it is convenient to use the following definitions:

$$SUCCEED \mathrel{\widehat{=}} \omega \to STOP, \qquad FAIL \mathrel{\widehat{=}} STOP.$$

We say that P may pass the test T, written P may T, if the parallel composition of P and T, hiding all events from Σ, can perform ω:

$$P \;\mathsf{may}\; T \mathrel{\widehat{=}} \exists\, tr \in \mathcal{T}[\![(P \underset{\Sigma}{\parallel} T) \setminus \Sigma]\!] \bullet \omega \text{ in } tr.$$

Note that the effect of hiding Σ is to make all these events urgent, corresponding to the intuition that if both P and T can do an event, then it happens immediately. Note also that the above definition could have used the semantic function $\mathcal{R}$ instead of $\mathcal{T}$: both models agree on the performance of events.

We say that two processes P and Q are timed testing equivalent if they pass precisely the same tests:

$$P \equiv_{may} Q \mathrel{\widehat{=}} (\forall T \in Test \bullet P \ \mathsf{may}\ T \Leftrightarrow Q \ \mathsf{may}\ T).$$

We now prove that timed testing equivalence indeed corresponds to equivalence in the timed testing model.

Theorem 6.1 $P \equiv_{may} Q \Leftrightarrow \mathcal{T}[\![P]\!] = \mathcal{T}[\![Q]\!]$.

Proof. For the right-to-left direction note that if $\mathcal{T}[\![P]\!] = \mathcal{T}[\![Q]\!]$ then for all tests T, $\mathcal{T}[\![(P \underset{\Sigma}{\parallel} T) \setminus \Sigma]\!] = \mathcal{T}[\![(Q \underset{\Sigma}{\parallel} T) \setminus \Sigma]\!]$, using the compositional semantics we gave earlier; hence $P\ \mathsf{may}\ T \Leftrightarrow Q\ \mathsf{may}\ T$.

For the left to right direction, we prove the contrapositive. Suppose $\mathcal{T}[\![P]\!] \neq \mathcal{T}[\![Q]\!]$. Then without loss of generality, there is some $tr \in \mathcal{T}[\![P]\!]$ such that $tr \notin \mathcal{T}[\![Q]\!]$. Let

$$\begin{aligned} tr = \langle & a_{11}, a_{12}, \ldots, a_{1m_1}, A_1, tock, \\ & a_{21}, a_{22}, \ldots, a_{2m_2}, A_2, tock, \\ & \ldots \\ & a_{n1}, a_{n2}, \ldots, a_{nm_n}\rangle. \end{aligned}$$

For convenience, we define a syntactic operator that attempts an event for one time unit, failing if it is not accepted:

$$a \xrightarrow{*} T \mathrel{\widehat{=}} (a \rightarrow T) \overset{1}{\rhd} FAIL.$$

We construct a test that succeeds when the process it is testing performs the trace tr, but fails, otherwise:

$$\begin{aligned} T \mathrel{\widehat{=}} & a_{11} \xrightarrow{*} a_{12} \xrightarrow{*} \ldots \xrightarrow{*} a_{1m_1} \xrightarrow{*} (?x : A_1 \rightarrow FAIL \\ & \overset{1}{\rhd} a_{21} \xrightarrow{*} a_{22} \xrightarrow{*} \ldots \xrightarrow{*} a_{2m_2} \xrightarrow{*} (?x : A_2 \rightarrow FAIL \\ & \overset{1}{\rhd} \ldots \\ & \overset{1}{\rhd} a_{n1} \xrightarrow{*} a_{n2} \xrightarrow{*} \ldots \xrightarrow{*} a_{nm_n} \xrightarrow{*} SUCCEED) \ldots). \end{aligned}$$

Note that the only way that T can reach the $SUCCEED$ state is if the process it is testing can perform the trace tr: clearly the processes need to perform the a_{ij} and $tock$ events in the appropriate order; and if the tested process can perform some x from A_i at the appropriate point, then that event will be performed (because it is hidden in the testing system), and so T will enter a $FAIL$ state; the only way the tested process can prevent this from happening is by refusing all of A_i. Hence $P\ \mathsf{may}\ T$ but not $Q\ \mathsf{may}\ T$. □

In untimed models, may-equivalence is the same as traces equivalence [1], i.e. with no refusal information; in continuous-time models, may-equivalence is the same as finite failures equivalence [11], i.e. with full refusal information; an interesting consequence of the above theorem is that may-equivalence in the discrete-time model lies between the may-equivalences in these other two scenarios.

6.2 No junk

In this section we show that every element of the semantic model $\mathcal{M}_{\mathcal{T}}$ is definable using the CSP syntax; i.e. the model contains no junk.

For $S \in \mathcal{M}_{\mathcal{T}}$, define the initial events, the initial maximum refusals of S, and the behaviours of S after event a and after $\langle A, tock\rangle$ as follows:

$$\begin{aligned} inits(S) &\mathrel{\widehat=} \{a \mid \langle a\rangle \in S\}, \\ initMaxRefs(S) &\mathrel{\widehat=} \{A \mid \langle A, tock\rangle \in S \wedge A \supseteq \Sigma - inits(S)\}, \\ S\ after\ a &\mathrel{\widehat=} \{tr \mid \langle a\rangle^\frown tr \in S\}, \\ S\ after\ \langle A, tock\rangle &\mathrel{\widehat=} \{tr \mid \langle A, tock\rangle^\frown tr \in S\}. \end{aligned}$$

Note that for $S \in \mathcal{M}_{\mathcal{T}}$:

- if $a \in inits(S)$ then $S\ after\ a \in \mathcal{M}_{\mathcal{T}}$;
- if $A \in initMaxRefs(S)$ then $S\ after\ \langle A, tock\rangle \in \mathcal{M}_{\mathcal{T}}$;
- if $\langle B, tock\rangle \in S$ then $\langle B \cup (\Sigma - inits(S)), tock\rangle \in S$ by repeated use of T3, and so $B \cup (\Sigma - inits(S)) \in initMaxRefs(S)$;
- $initMaxRefs(S)$ contains $\Sigma - inits(S)$: this follows from the previous item because $\langle \{\}, tock\rangle \in S$ by T1 and T4.

Given $S \in \mathcal{M}_{\mathcal{T}}$, we now define a process $P(S)$ that has precisely the behaviours in S:

$$P(S) \mathrel{\widehat=} \bigsqcap_{A : initMaxRefs(S)} \begin{pmatrix} ?x : \Sigma - A \rightarrow P(S\ after\ x) \\ \quad \overset{1}{\rhd} P(S\ after\ \langle A, tock\rangle). \end{pmatrix}$$

(We note that $P(S)$ may not be finitely expressible, because of the indexing by potentially non-constructive sets; we are not considering effective computability here.)

Lemma 6.2 $\forall S \in \mathcal{M}_{\mathcal{T}} \bullet \mathcal{T}[\![P(S)]\!] = S$.

Proof. We begin by showing that for every timed test tr, for every $S \in \mathcal{M}_{\mathcal{T}}$, if $tr \in S$ then $tr \in \mathcal{T}[\![P(S)]\!]$. The proof is by induction on the length of tr. The base case is trivial. So suppose $tr \in S$ is non-empty, and perform a case analysis as follows:

- Case $tr = \langle a\rangle^\frown tr'$. Then $a \in inits(S)$ and $tr' \in S\ after\ a$; so by the inductive hypothesis, $tr' \in \mathcal{T}[\![P(S\ after\ a)]\!]$. Hence, letting $A \mathrel{\widehat=} \Sigma - inits(S)$, we have $a \in \Sigma - A$ and so $tr \in \mathcal{T}[\![?x : \Sigma - A \rightarrow P(S\ after\ x)]\!]$, and so $tr \in \mathcal{T}[\![P(S)]\!]$ because $A \in initMaxRefs(S)$.
- Case $tr = \langle B, tock\rangle^\frown tr'$. Let $A \mathrel{\widehat=} B \cup (\Sigma - inits(S))$. Now, by repeated use of T3, $\langle A, tock\rangle^\frown tr' \in S$, so $A \in initMaxRefs(S)$. Hence $tr' \in S\ after$

$\langle A, tock \rangle$, and so by the inductive hypothesis $tr' \in \mathcal{T}[\![P(S\ after\ \langle A, tock \rangle)]\!]$. Hence $tr \in \mathcal{T}[\![?x : \Sigma - A \to P(S\ after\ x) \overset{1}{\triangleright} P(S\ after\ \langle A, tock \rangle)]\!]$ and so $tr \in \mathcal{T}[\![P(S)]\!]$.

We now show the converse: that if $tr \in \mathcal{T}[\![P(S)]\!]$ then $tr \in S$, for every $S \in \mathcal{M}_\mathcal{T}$. The proof is by induction on the length of tr. The base case is again trivial. So suppose $tr \in \mathcal{T}[\![P(S)]\!]$ is non-empty, and perform a case analysis over the form of tr:

- Case $tr = \langle a \rangle ^\frown tr'$. Then for some $A \in initMaxRefs(S)$, $a \in \Sigma - A$ (so $a \in inits(S)$), and $tr' \in \mathcal{T}[\![P(S\ after\ a)]\!]$. So by the inductive hypothesis, $tr' \in S\ after\, a$ so $tr \in S$.
- Case $tr = \langle B, tock \rangle ^\frown tr'$. Then for some $A \in initMaxRefs(S)$, we have $B \subseteq A$ and $tr' \in \mathcal{T}[\![P(S\ after\ \langle A, tock \rangle)]\!]$. Hence by the inductive hypothesis, $tr' \in S\ after\ \langle A, tock \rangle$, and so $\langle A, tock \rangle ^\frown tr' \in S$. But then $tr = \langle B, tock \rangle ^\frown tr' \in S$, because S satisfies T2.

Finally, we consider the time-guardedness of the $P(S)$ recursions. The definition as it is written is not obviously time-guarded: however, we can transform it into a form where it is. Replace each recursive call of the form $P(S\ after\ x)$ by the right-hand-side of the corresponding process definition; keep repeating this replacement. Now, S satisfies T5, so there is some upper bound n on the number of events that can be performed before the first *tock*; hence the above replacement can be repeated at most n times before all branches meet processes $P(S')$ with $inits(S') = \{\}$. Then all recursive calls are on the right-hand-side of a timeout, so the definition of $P(S)$ is time guarded in all variables. □

Hence the semantic model $\mathcal{M}_\mathcal{T}$ satisfies the no junk condition:

Theorem 6.3 *For every $S \in \mathcal{M}_\mathcal{T}$, there is a process P such that $\mathcal{T}[\![P]\!] = S$.*

6.3 Full abstraction

The results of this section may be summarised in the following result.

Corollary 6.4 *The timed testing model is fully abstract with respect to the test "can perform ω".*

7 Traces

In this section, we consider a yet more abstract model for discrete-time CSP, namely a pure traces model, containing no refusal information. The semantics of a process can be defined by projecting from the timed tests:

$$\mathcal{T}r[\![P]\!] = \{trace(tr) \mid tr \in \mathcal{T}[\![P]\!]\}.$$

(Of course, the traces could also have been obtained by projecting from the refusal traces.) We will say that two processes are *trace equivalent* if they have the same traces.

Our main reason for considering this model is that we want to make a connection between our definition of the quantity of information flow from [4], and Shannon information flow theory [12]. In the latter, one considers a probabilistically unreliable channel, say between A and B. Further, the channel is *generative* [14]: it is always willing to output a single value to B (that value depending probabilistically on what was input by A). This means that B receives no extra information by seeing refusals: if the channel is willing to output y, then he gains the same information by seeing the rest of his alphabet refused as by performing y itself. For this reason, it seems that B's observational power in this setting corresponds to the traces model.

It is well known that this model is not a congruence. For example, let

$$P = a \rightarrow (b \rightarrow STOP \overset{1}{\rhd} STOP), \qquad P' = P \sqcap Wait\,2\,; P.$$

Then $\mathcal{T}r[\![P]\!] = \mathcal{T}r[\![P']\!]$: both are given by the regular expression $tock^* + tock^*.a.tock^* + tock^*.a.b.tock^*$. But $\mathcal{T}r[\![P \setminus \{a\}]\!] \neq \mathcal{T}r[\![P' \setminus \{a\}]\!]$: the latter includes $\langle tock, tock, b\rangle$, whereas in the former process the a will be performed (silently) before the first $tock$, and so the b can only be performed before the first $tock$. However, this model is a congruence with respect to all operators other than hiding.

7.1 Testing

We now define an alternative notion of timed may testing. Note that in the untimed setting, the standard definition of P may T:

$$\langle\omega\rangle \in traces((P \underset{\Sigma}{\parallel} T) \setminus \Sigma)$$

is equivalent to

$$\exists\, tr \in traces(P \underset{\Sigma}{\parallel} T) \bullet \omega \text{ in } tr.$$

When one lifts these two definitions to a timed setting, they turn out to be different: the former corresponds to the definition from Section 6; the latter corresponds to the definition below.

We write P may' T if test T, in parallel with P —without hiding— can perform the event ω:

$$P \;\mathsf{may}'\; T \Leftrightarrow \exists\, tr \in \mathcal{T}[\![P \underset{\Sigma}{\parallel} T]\!] \bullet \omega \text{ in } tr.$$

We then say that P and Q are may' equivalent if they pass the same tests:

$$P \equiv_{\mathsf{may}'} Q \Leftrightarrow \forall\, T \in Test \bullet P \;\mathsf{may}'\; T \Leftrightarrow Q \;\mathsf{may}'\; T.$$

Note this is different from the definition of may. Consider

$$P = a \rightarrow STOP, \qquad T = (a \rightarrow FAIL) \overset{1}{\rhd} SUCCEED.$$

Then P may' T (on trace $\langle\{\}, tock, \omega\rangle$), but not P may T (since the hiding ensures that the a happens before time 1). By contrast, define

$Q = a \to STOP \sqcap STOP.$

Then Q may T and Q may' T. So may testing distinguishes P and Q; however, the above test does not distinguish P and Q under may' testing, and, in fact, no other does, as confirmed by the following theorem.

Theorem 7.1 $P \equiv_{may'} Q \Leftrightarrow \mathcal{T}r[\![P]\!] = \mathcal{T}r[\![Q]\!]$.

Proof. For the left to right implication, we prove the contrapositive. Suppose P and Q do not have the same traces. Then without loss of generality, suppose

$$\begin{aligned} tr = \langle & a_{11}, a_{12}, \ldots, a_{1m_1}, tock, \\ & a_{21}, a_{22}, \ldots, a_{2m_2}, tock, \\ & \ldots \\ & a_{n1}, a_{n2}, \ldots, a_{nm_n} \rangle \end{aligned}$$

is a trace of P but not of Q. We construct a test that succeeds when the process it is testing performs the trace tr, but fails, otherwise:

$$\begin{aligned} T \mathrel{\widehat{=}} & a_{11} \xrightarrow{*} a_{12} \xrightarrow{*} \ldots \xrightarrow{*} a_{1m_1} \xrightarrow{*} Wait\, 1; \\ & a_{21} \xrightarrow{*} a_{22} \xrightarrow{*} \ldots \xrightarrow{*} a_{2m_2} \xrightarrow{*} Wait\, 1; \\ & \ldots \\ & a_{n1} \xrightarrow{*} a_{n2} \xrightarrow{*} \ldots \xrightarrow{*} a_{nm_n} \xrightarrow{*} SUCCEED. \end{aligned}$$

For the reverse direction, it is easy to show that

$$\mathcal{T}r[\![P \underset{A}{\parallel} T]\!] = \{tr \mid \exists\, tr_P \in \mathcal{T}r[\![P]\!], tr_T \in \mathcal{T}r[\![T]\!] \bullet tr \in tr_P \underset{A}{\parallel} tr_T\},$$

where the parallel composition of traces is similar to as defined in Figure 1. Hence if $\mathcal{T}r[\![P]\!] = \mathcal{T}r[\![Q]\!]$, then $\mathcal{T}r[\![P \underset{\Sigma}{\parallel} T]\!] = \mathcal{T}r[\![Q \underset{A}{\parallel} T]\!]$, so P may' $T \Leftrightarrow Q$ may' T. $\square$

8 Conclusions

In this paper we have studied a compositional denotational model for Timed CSP, which we consider to be simpler than existing models. Further, we have shown that equivalence in the model corresponds to timed may testing, and hence this is the coarsest reasonable model of the language. Further, we have shown that the model contains no junk, and so precisely captures the properties of expressible processes. Finally, we have considered a weaker model based on traces.

In this section, we briefly discuss some consequences of our decision to introduce a strict timeout operator to the language; we then discuss some related and future work.

8.1 On strict timeouts

As noted earlier, the strict timeout operator seems to have a far more natural semantics than the weak timeout; in particular, is seems necessary to model some natural examples, such as the example from Section 1.

The correspondence with timed may testing does not depend on the strict timeout. The construction in Section 6.1 can be adapted to work with a weak timeout, by replacing the definition of $a \xrightarrow{*} T$ by

$$a \xrightarrow{*} T \mathrel{\widehat{=}} (a \rightarrow T) \overset{0}{\triangleright} FAIL,$$

i.e. the a is available initially, but will be withdrawn sometime before the first *tock*. The reader may check that the rest of the proof goes through as before.

However, without the strict timeout, the model of this paper does not satisfy the no junk condition. It can be shown that, without this strict timeout, all processes satisfy the condition

(1) $\quad tr^\frown\langle A, tock\rangle \in S \wedge tr^\frown\langle A \cup \{a\}, tock\rangle \notin S \Rightarrow tr^\frown\langle A, tock, a\rangle \in S,$

which says that if an event is not refused before a *tock* (so must be available), then it will still be available after a *tock*. Therefore, any semantic object that does not satisfy the above condition cannot be expressed syntactically without the strict timeout.

It is interesting to ask whether adding the above condition (1) as an axiom [7], and still omitting the strict timeout, gives us no junk. We believe not. Consider the process that initially offers an a, and if that a is performed after n *tock*s then offers a b after a further n *tock*s. This can be modelled using the strict timeout by

$$P(0), \text{ where } P(n) = (a \rightarrow Wait\,n\,; b \rightarrow STOP) \overset{1}{\triangleright} P(n+1).$$

This process satisfies the extra axiom. However, we believe that it is not expressible without the strict timeout, for one cannot detect the precise time at which the a occurs.

In [6,7], it is shown that the discrete-time models satisfy a very close relationship with the continuous-time models of Timed CSP from [9]. It would be interesting to consider whether the continuous-time models can be extended with a strict timeout, and whether the same relationship would then hold with the model of this paper.

8.2 Related work

In [3], a fully abstract denotational semantics is presented for an OCCAM-like language. The authors prove that processes distinguished by the denotational model are also distinguished by a criterion extracted from the operational semantics; we consider this a less natural criterion than may testing. (They do not consider the absence of junk.) Like us, they consider failures (actually convex closure of ready sets, which is equivalent). They consider a language with synchronous computation, and thus do not consider the relative order of events within the same time unit: hence the question of refusals between

[7] This condition is similar to the axiom from [6] discussed in footnote 6.

non-*tock* events —the difference between the refusal traces and timed testing models— does not arise.

In [7], a different kind of full abstraction result is proven. It is shown that if two processes are distinguished by the model, then there is a natural (i.e., closed under inverse digitisation) specification ϕ, such that one process satisfies ϕ and the other doesn't.

8.3 Future work

As noted above, our main reason for considering the model of this paper was our interest in quantifying information flow in multi-level secure systems. We want to continue this work. In particular, as noted in Section 7, we want to relate our definition to Shannon information flow, either using the timed testing model or the traces model. Further, we want to consider algorithms for calculating the quantity of information flow of a given process, and, indeed, whether the problem is decidable in general.

Acknowledgements

We would like to thank Irek Ulidowski and the anonymous referees for useful comments about this work.

References

[1] R. de Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[2] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[3] C. Huizing, R. Gerth, and W. P. de Roever. Full abstraction of a real-time denotational semantics for an occam-like language. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 223–236, 1987.

[4] G. Lowe. Defining information flow quantity. *Journal of Computer Security*, 12(3, 4):619–653, 2004.

[5] A. Mukarram. *A Refusal Testing Model for CSP*. D. Phil thesis, Oxford, 1993.

[6] J. Ouaknine. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems*. D.Phil thesis, Oxford University, 2000.

[7] J. Ouaknine. Digitisation and full abstraction for dense-time model checking. In *Proceedings of TACAS*, 2002.

[8] I. Phillips. Refusal testing. *Theoretical Computer Science*, 1987.

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

[10] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[11] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 1999.

[12] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1963.

[13] W. A. Sutherland. *Introduction to Metric and Topological Spaces.* Oxford University Press, 1983.

[14] R. J. van Glabbeek, S. A. Smolka, B. Steffen, and C. Tofts. Reactive, generative and stratified models of probabilistic processes. In *IEEE Symposium on Logic in Computer Science*, 1990.